



# **Dialogic® NaturalAccess™ Fusion VoIP API Developer's Manual**

## Copyright and legal notices

---

Copyright © 2009-2014 Dialogic Inc. All Rights Reserved. You may not reproduce this document in whole or in part without permission in writing from Dialogic Inc. at the address provided below.

All contents of this document are furnished for informational use only and are subject to change without notice and do not represent a commitment on the part of Dialogic Inc. and its affiliates or 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 certain safety-affecting situations. Please see <http://www.dialogic.com/company/terms-of-use.aspx> for more details.

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 Inc. at the address indicated below or on the web at [www.dialogic.com](http://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 6700 de la Cote-de-Liesse Road, Suite 100, Borough of Saint-Laurent, Montreal, Quebec, Canada H4T 2B5. **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.**

Dialogic, Dialogic Pro, Dialogic Blue, Veraz, Brooktrout, Diva, BorderNet, PowerMedia, ControlSwitch, I-Gate, Mobile Experience Matters, Network Fuel, Video is the New Voice, Making Innovation Thrive, Diastar, Cantata, TruFax, SwitchKit, Eiconcard, NMS Communications, SIPcontrol, Exnet, EXS, Vision, inCloud9, NaturalAccess and Shiva, among others as well as related logos, are either registered trademarks or trademarks of Dialogic Inc. and its affiliates or 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 6700 de la Cote-de-Liesse Road, Suite 100, Borough of Saint-Laurent, Montreal, Quebec, Canada H4T 2B5. 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.

The names of actual companies and products 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
9000-6752-10	November, 1999	MVH, Fusion 4.0 preliminary release
9000-6752-11	February, 2000	MVH, Fusion 4.0 beta
9000-6752-12	July, 2000	MVH, Fusion 4.0 beta
9000-6752-13	October, 2000	MVH, Fusion 4.0 service pack
9000-6752-14	February, 2001	MVH, Fusion 4.1
9000-6752-15	June, 2001	MVH, Fusion 4.2 beta
9000-6752-16	October, 2001	MVH, Fusion 4.2
9000-6752-17	July, 2002	MVH, Fusion 4.2.1
9000-6752-18	February, 2003	MVH, Fusion 4.3 beta
9000-6752-19	April, 2003	MVH, Fusion 4.3
9000-6752-20	December, 2003	MVH, Fusion 4.4
9000-6752-21	December, 2004	MVH, Natural Access 2005-1 beta
9000-6752-22	February, 2005	MVH, Natural Access 2005-1
9000-6752-23	October, 2005	MVH, Natural Access 2005-1, SP 1
9000-6752-24	December 2006	LBZ, Natural Access 2005-1, SP 3
9000-6752-25	October 2007	DEH, Natural Access 2005-1, SP 5
9000-6752-26	June 2008	LBZ, Natural Access R8
64-0497-01	October, 2009	LBG, NaturalAccess R9.0
64-0497-02	December, 2009	LBG, NaturalAccess R9.0.1
64-0497-03	October, 2014	PKN, NaturalAccess R9.04 update
Last modified: October 24, 2014		

Refer to [www.dialogic.com](http://www.dialogic.com) for product updates and for information about support policies, warranty information, and service offerings.



# Table Of Contents

<b>Chapter 1: Introduction .....</b>	<b>9</b>
<b>Chapter 2: Terminology .....</b>	<b>11</b>
<b>Chapter 3: Overview of Fusion .....</b>	<b>13</b>
Fusion defined .....	13
Fusion media server applications .....	14
Fusion requirements .....	15
CG boards .....	15
Universal ports .....	15
Resource management .....	16
NaturalAccess .....	17
NaturalAccess features .....	18
Contexts and event queues .....	18
NaturalAccess APIs .....	18
NaturalAccess OAM API .....	18
MSPP API .....	19
MSPP API programming objects .....	19
MSPP API connections .....	20
Initiating MSPP media connections .....	21
Available channels .....	21
Available endpoints .....	23
MSPP filters .....	23
<b>Chapter 4: Getting started with Fusion .....</b>	<b>25</b>
Before using Fusion .....	25
Getting started tasks .....	25
Configuring CG boards .....	26
Booting CG boards .....	27
Verifying the installation .....	28
Using CG board utilities .....	28
Using msppsamp .....	29
Compiling and linking .....	30
<b>Chapter 5: Customizing the Fusion configuration .....</b>	<b>31</b>
Customizing the board configuration .....	31
Configuring CG board resources .....	31
DSPs .....	31
Using f54info .....	32
Using DSP and Resource keywords .....	32
Resource keywords .....	33
Media and call progress masks .....	35
Configuring CG board Ethernet interfaces .....	36
Setting Ethernet interfaces to use the IPv4 stack .....	36
Setting Ethernet interfaces to use the IPv6 stack .....	37
Setting up the board to run in dual IPv4 and IPv6 stack mode .....	38
Sample E1 system configuration file .....	39
<b>Chapter 6: Managing MSPP connections .....</b>	<b>43</b>
Creating NaturalAccess contexts and queues .....	43

Opening MSPP service instances .....	43
Setting up MSPP connections .....	44
Tearing down MSPP connections .....	45
Sending MSPP queries and commands .....	45
Filter queries and commands .....	45
System-level queries .....	46
MSPP function sequence .....	47
Connecting MSPP channels during call set up .....	49
Receiving inbound calls .....	50
Placing outbound calls .....	52
<b>Chapter 7: Using RTP forking and switching .....</b>	<b>55</b>
RTP forking and RTP switching .....	55
Calling card service scenario .....	56
Monitoring application scenario .....	58
RTP switching limitations .....	60
<b>Chapter 8: Looping back voice data .....</b>	<b>61</b>
Loopback connections .....	61
DS0 loopback connections .....	61
RTP loopback connections .....	62
Simplex connections .....	62
Duplex connections .....	62
<b>Chapter 9: Processing MSPP service unsolicited events .....</b>	<b>65</b>
Overview of unsolicited events .....	65
RTP and UDP endpoint route availability events .....	65
Types of unsolicited events .....	66
RTCP report events .....	67
RTCP messages .....	67
Configuring RTP IPv4 and IPv6 endpoints to return RTCP report information ....	68
Processing RTCP information .....	68
RFC 2833 related events .....	69
<b>Chapter 10: Eliciting board information through MSPP queries .....</b>	<b>71</b>
Board-level queries and unsolicited events .....	71
Ethernet link status queries .....	72
CPU utilization queries .....	72
Route availability events .....	73
<b>Chapter 11: Implementing RFC 2833 support .....</b>	<b>75</b>
Transferring DTMF digits according to RFC 2833 .....	75
Transferring DTMF digits with RFC 2833 .....	75
Transferring non-DTMF RFC 2833 events .....	77
Setting up RFC 2833 capabilities .....	77
Configuring channels for RFC 2833 support .....	79
Customizing RFC 2833 compliant channel behavior .....	80
Setting encoder RFC 2833 parameters .....	80
Setting decoder RFC 2833 parameters .....	81
Playing tones manually through a voice decoder filter .....	81
Configuring RFC 2833 settings .....	82
Customizing RTP endpoint behavior .....	83
Configuring RTP endpoint parameters .....	83

Sending RTP IPv4 and IPv6 endpoint filter commands .....	83
<b>Chapter 12: Using native play and record .....</b>	<b>85</b>
Performing NMS native play and record .....	85
NMS native play and record advantages .....	85
Implementing NMS native play and record .....	85
Native play .....	86
Sample procedure .....	86
Example .....	88
Native record with inband silence and DTMF detection .....	88
Sample procedure .....	89
Native record without inband silence and DTMF detection .....	92
Sample procedure .....	93
<b>Chapter 13: Implementing DCE over IP transport .....</b>	<b>97</b>
Fusion DCE transport .....	97
Emitting side.....	97
Receiving side.....	99
Setting up DCE transport capability .....	100
Configuring the echo canceller .....	101
Configuring clear G.711 channels .....	102
<b>Chapter 14: Using T.38 fax connections .....</b>	<b>103</b>
T.38 fax connections overview .....	103
T.38 fax transmission model .....	103
T.38 fax relay.....	104
T.38 fax standard .....	104
Fusion T.38 fax channels .....	104
Using T.38 fax full duplex channels.....	105
Setting up fax and voice connections .....	107
Connection setup tasks .....	108
Performing voice and fax switchover .....	109
Receiving gateway .....	109
Emitting gateway.....	110
Switching from voice to fax .....	112
Receiving fax billing events.....	113
<b>Chapter 15: Using ThroughPacket multiplexing .....</b>	<b>115</b>
ThroughPacket multiplexing overview .....	115
ThroughPacket (TPKT) endpoints.....	116
Increasing data rate efficiency .....	117
Reducing packet rates.....	118
Implementing ThroughPacket multiplexing .....	119
Using ThroughPacket OAM board keywords.....	120
ThroughPacket resource (DLM) file .....	120
TPKT data transmission parameters .....	121
Example .....	122
Creating TPKT endpoints .....	123
Specifying TPKT endpoint parameters .....	124
TPKT endpoint parameters.....	124
Session IDs.....	124
Session sequence flags.....	125
Connecting TPKT endpoints.....	126

Configuring G.723.1 vocoders for use with TPKT endpoints .....	126
Sending commands to TPKT endpoints .....	127
Sending queries to TPKT endpoints .....	128
<b>Chapter 16: RTP MIB extended management component (EMC) .....</b>	<b>129</b>
RTP MIB EMC .....	129
Using the RTP MIB .....	129
Using structured values .....	130
Controlling events.....	131
Using OAM API functions with the RTP EMC.....	131
RTP EMC object hierarchy .....	132
Object summary.....	134
RTP EMC keyword qualifiers .....	136
RTP EMC keyword qualifiers .....	136
RTP EMC keyword qualifiers .....	136
<b>Chapter 17: Demonstration programs .....</b>	<b>137</b>
MSPP exerciser: msppxsr .....	137
Fusion nailed-up sample: msppsamp .....	141
msppsamp overview .....	141
Running msppsamp .....	142
Details of operation.....	147
RTP switch sample: rtpswitchsamp .....	148
Native play and record test: natprdemo .....	151
Native play and record test: natprtest.....	153
Programming notes.....	159
ThroughPacket sample: tpktsamp.....	160
Running tpktsamp .....	160
<b>Chapter 18: Fusion migration .....</b>	<b>163</b>
Migrating from Fusion 3 to Fusion 4 .....	163
Migrating to Fusion 4.4 .....	165
Channel enhancements .....	165
Configurable defaults .....	166
Other Fusion 4.4 Features .....	167



---

# 1 Introduction

---

The *Dialogic® NaturalAccess™ Fusion VoIP API Developer's Manual* provides an overview of the functions and capabilities of NaturalAccess Fusion VoIP API software and hardware components. The manual also provides information about the Fusion API configurations and programming models.

This manual is for developers of IP telephony gateway applications who are using NaturalAccess and the MSPP (Media Stream Protocol Processing) API. This document defines telephony terms where applicable, but assumes that you are familiar with IP telephony concepts, circuit switching terminology, and the C programming language.

Use this manual in conjunction with the *Dialogic® NaturalAccess™ Media Stream Protocol Processing API Developer's Manual* and the CG board documentation.



## 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
Point-to-Point Switching Service	Dialogic® NaturalAccess™ Point-to-Point Switching API
Switching Service	Dialogic® NaturalAccess™ Switching Interface API

Former terminology	Dialogic terminology
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

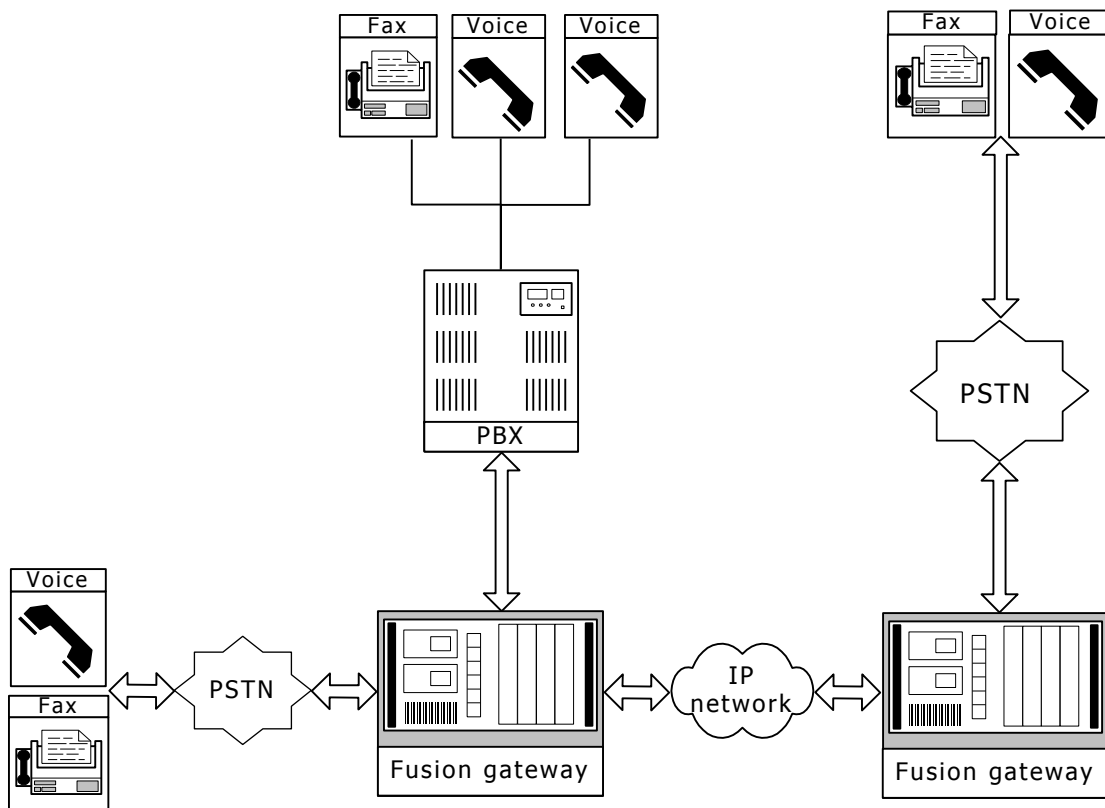
## Overview of Fusion

### Fusion defined

The NaturalAccess Fusion VoIP API consists of hardware and software for producing IP telephony gateway applications and media server applications.

Gateway applications provide a way of transferring data between endpoints created at telephone network and IP network interfaces. These applications convert voice and fax data as they transfer it from network to network. For example, when a gateway transfers voice data between a PSTN and an IP network, it converts the data between PCM and data packet form and (when applicable) encodes and compresses the data. The data can then be switched over traditional telephone networks, sent over IP networks to compatible computer client terminals, or directed to other gateways.

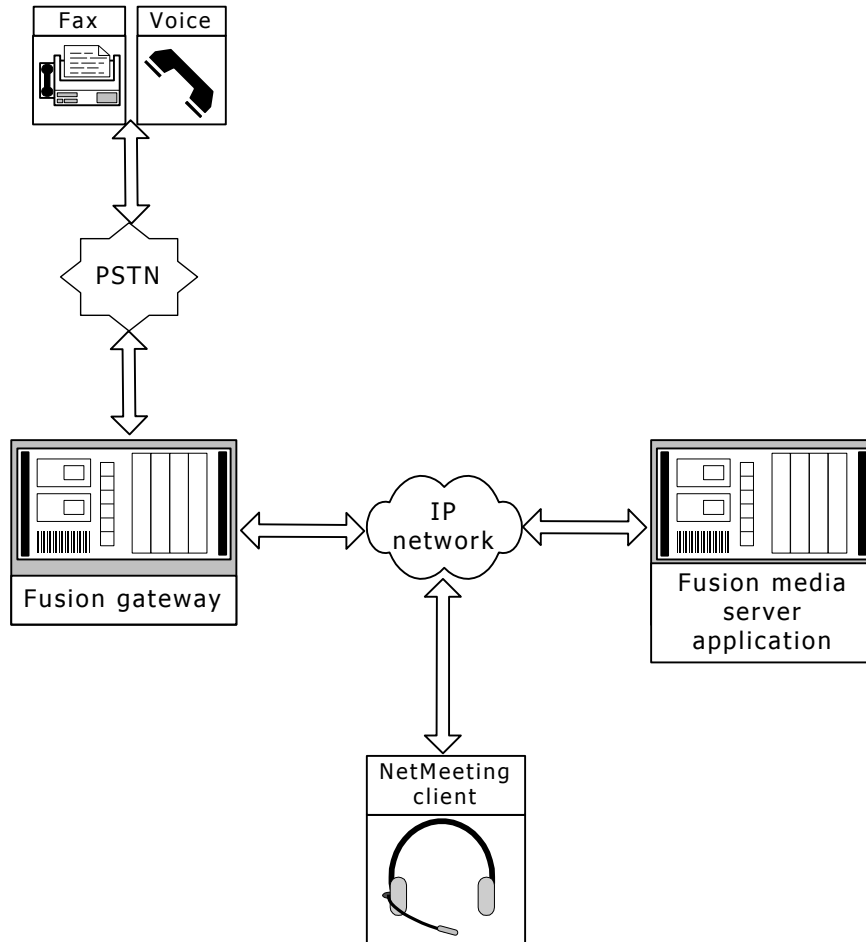
The following illustration shows two Fusion gateway systems transferring voice and fax data from one PSTN, over an IP network, to another PSTN:



## Fusion media server applications

Fusion media server applications perform media processing and IVR tasks such as providing voice prompts, detecting and generating DTMF digits, or using speech recognition to query or deliver information. Fusion media server systems do not need to be connected to a PSTN and may instead only receive voice data from an IP network (through a Fusion gateway system or directly from a client application such as NetMeeting).

The following illustration shows a Fusion media server receiving voice data over an IP network through a Fusion gateway system:



## Fusion requirements

---

Fusion includes a set of software APIs that enable applications to link PSTN calls or IVR functions (carrying PCM data) to IP sessions (carrying packet data), and control the flow of data between PSTN and IP networks. Fusion requires the following hardware and software:

- CG board
- NaturalAccess
- Fusion VoIP API software

Fusion configurations use CG boards to receive and transmit data to PSTN and IP networks. They can use Natural Call Control functions to place and receive PSTN calls, and the MSPP service to create connections for transmitting data to the IP network. Fusion does not provide an API for performing IP call control. To perform IP call control, applications can use a third party call control stack such as H.323.

## CG boards

---

CG series boards provide the following features:

- Up to 16 T1 or E1 (75 or 120 ohm) network interfaces for digital trunk connectivity through a rear transition board (you must configure the board for T1 or E1 operation).
- Full support of the H.110 bus specification, which allows boards to share data with other boards on the H.110 bus.
- Telephony bus switching for a total of 256 full duplex connections between local devices and the H.110 bus. Switch connections are allowed between local devices and are non-blocking.
- Two 10/100 Base-T Ethernet connections for Fast Ethernet connectivity to IP networks. CG board Ethernet interfaces can be configured to use an on-board IPv4 stack, to use an on-board IPv6 stack, or to run in dual IPv4 and IPv6 mode.
- Up to 96 high-performance digital signal processor (DSP) cores that provide resources for universal IP gateway ports. The functionality provided by these processors depends on the resource management configured for the board.

The port densities supported by specific Fusion configurations vary according to the number and types of DSP resources the system requires.

## Universal ports

---

CG boards can support up to 360 simultaneous universal ports of voice and fax data. A universal port can use any combination of loaded functions, such as voice play, voice record, tone detection, tone generation, T.38 fax, or voice encode and decode functions. Applications can also switch between these functions within a single phone call.

A universal port consists of two full duplex complex vocoders, one simple vocoder (simple vocoders use algorithms that achieve a lower level of compression and require less processing resources than complex vocoders.), T.38 fax, echo cancellation, and various call control functions such as DTMF detection, and cleardown tone detection. G.711 is the only available simple vocoder. All other vocoders (G.729A, G.723.1, G.726) are considered complex. By definition, a universal port runs on a single DSP core (there are two DSP cores per DSP chip).

Specific universal port capabilities are determined by the resource management setup for the CG board and universal port densities can vary based on the number of T.38 fax ports running on the system. For more information about resource management, refer to the CG board documentation. For more information about port densities with T.38 fax, refer to the *readmefusion.txt* file.

For Fusion applications, each universal port can support the following resources:

- Up to two complex voice vocoders (vocoders that perform encoding and decoding for voice algorithms such as G.723.1, G.729A, and G.726)
- One simple voice vocoder (such as G.711)
- T.38 fax processing
- PSTN call control processing

Universal ports enable the application to process voice and fax data during the course of a single call, without being concerned with the board resources needed to perform the processing activities. However, when vocoders are loaded, only one encoder/decoder pair can be active at any given time per channel.

## Resource management

---

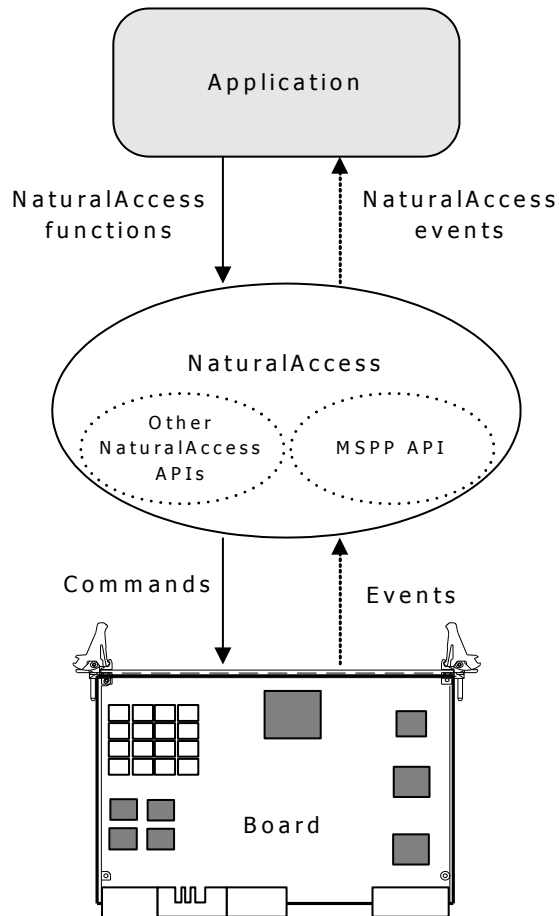
CG boards use an on-board resource manager that allocates and loads board DSP resources based on configuration information specified through a board keyword configuration file.

For detailed information about CG board resource management, refer to the CG board documentation.



## NaturalAccess

Fusion gateway applications use the NaturalAccess development environment for performing voice and call processing operations on NaturalAccess boards. The following illustration shows an overview of NaturalAccess components:



This topic provides information about:

- NaturalAccess features
- Contexts and event queues
- NaturalAccess APIs
- NaturalAccess OAM API

## NaturalAccess features

---

NaturalAccess provides:

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

## Contexts and event queues

---

In the NaturalAccess development environment, a context organizes services and accompanying resources around a single processing context. A context usually represents an application instance controlling a single telephone call.

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

## NaturalAccess APIs

---

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

Fusion software includes the MSPP API for creating and controlling media connections between PSTNs and IP networks. For information about the MSPP API, refer to the *Dialogic® NaturalAccess™ Media Stream Protocol Processing API Developer's Manual*.

## NaturalAccess OAM API

---

The NaturalAccess OAM API provides a mechanism for applications to configure and manage board resources in gateway systems. Use the OAM API to configure and initialize the hardware in the chassis by managing:

- Hardware components, such as AG or CG Series boards.
- Software components, such as the Hot Swap process.

Using the OAM API, you can:

- Create, delete, and query the configuration of a board managed component.
- Start (boot), stop (shut down), and test a managed component.
- Receive notifications from managed components.

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

## MSPP API

The MSPP API provides functions for building unified voice and fax gateway applications by creating, connecting, configuring, and destroying media endpoints and channels. Applications can create, connect, and enable duplex or simplex media connections by managing objects called endpoints and channels. Endpoints provide nodes for sending and receiving network data. Channels provide a mechanism for converting and transferring data as it flows from one endpoint to another. Use the MSPP API in combination with other Natural Access APIs, such as the NaturalCallControl (NCC) and ADI APIs, to provide an interface for PSTN call control and IVR.

The MSPP API does not perform call control at either the PSTN or packet network interfaces. Applications must perform call control independently by using functions from the NCC API for PSTN-side call control and a third party call control stack (such as SIP) for IP-side call control. For more information about using the NaturalCallControl API, refer to the *Dialogic® NaturalAccess™ NaturalCallControl™ API Developer's Manual*.

This topic provides information about the following:

- MSPP API programming objects
- MSPP API connections
- Initiating MSPP media connections

For more information, refer to the *Dialogic® NaturalAccess™ Media Stream Protocol Processing API Developer's Manual*.

### MSPP API programming objects

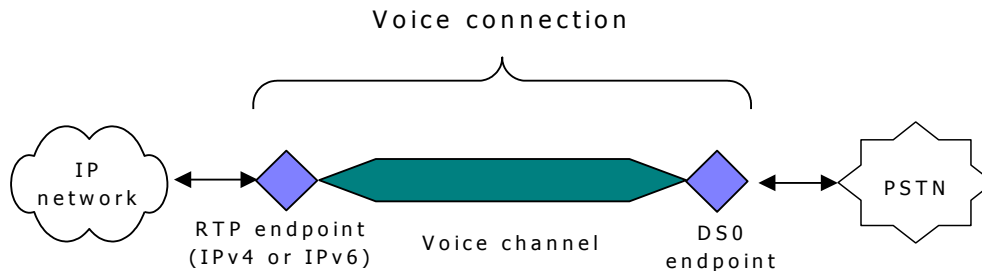
Applications use the MSPP API to create and control the following components:

Term	Description
endpoint	An entry and/or exit point for data that streams through the gateway through an MSPP channel.  For example, a DS0 endpoint provides the terminus through which PCM data travels to and from a PSTN network. An RTP endpoint provides a node through which packet data (framed in RTP format) travels to and from an IP network.
channel	A linked set of functions that transform a real-time flow of voice or fax data from one form to another. The flow of data can either be two-way (duplex) or one-way (simplex).  For example, a G.711 full-duplex channel provides all the necessary processing for G.711 data to flow between a PSTN network and an IP network.
connection	A direct coupling between MSPP endpoints and a channel that transfers and processes voice or fax data. When the connection is enabled, data flows between the endpoints.

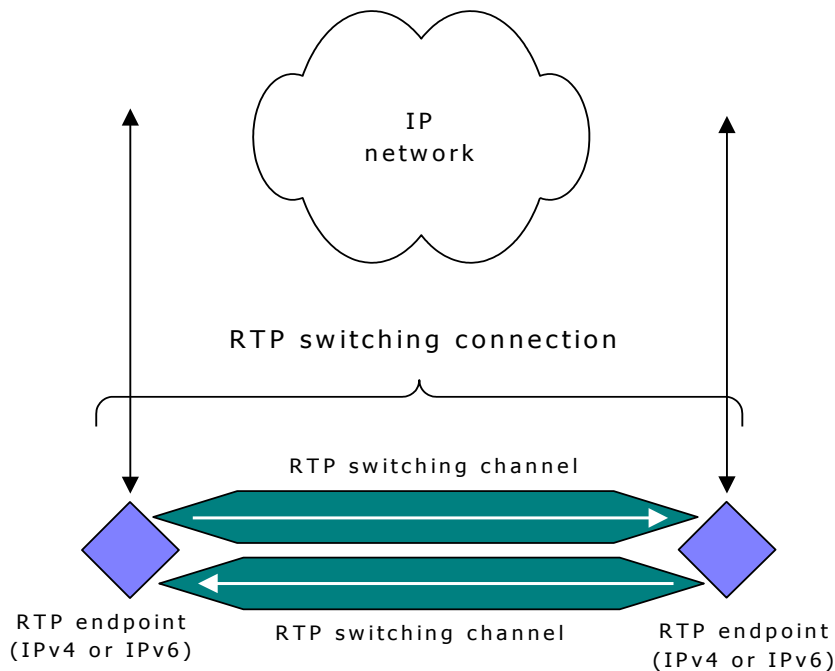
For example, an application can create a typical Fusion MSPP connection by joining a DS0 endpoint at the PSTN interface, and an RTP endpoint (IPv4 or IPv6) at the IP network interface, by means of a full duplex G.711 voice channel. Because the channel is full duplex, it performs G.711 encoding or decoding depending on the direction the data is moving in. When the connection is enabled, it forms an active, full duplex voice path across the gateway from network to network.

## MSPP API connections

Fusion allows applications to create several different kinds of channels. The following illustration shows the components that make up a typical MSPP voice connection. In this case, the connection is used for transferring and processing voice data between a PSTN and IP network:



Applications can also use RTP switching channels to route voice data to more than one RTP endpoint. This mechanism allows applications to monitor the voice data transferred across a particular voice channel. The following illustration shows the components used to connect voice data between two RTP endpoints. For more information about how to implement RTP switching, refer to *RTP forking and RTP switching* on page 55. Because MSPP switch channels are always simplex (whereas voice channels can be simplex or duplex), applications must connect two switch channels to enable a full duplex switch connection between two duplex RTP endpoints. The following illustration shows a pair of RTP switching channels used to transfer a duplex audio stream:



## Initiating MSPP media connections

Applications create media channels across the Fusion gateway in the following way:

- Creating MSPP endpoints and MSPP channels. Applications create endpoints and channels independently and in any sequence.
- Connecting a pair of MSPP endpoints with a channel to form an MSPP connection.
- Enabling the channel so that data begins to flow through the media connection.

The application performs PSTN and IP call control independently at the PSTN and IP interface. For PSTN call control the application can use the NCC service. For more information about PSTN call control, refer to *Receiving inbound calls* on page 50 and *Placing outbound calls* on page 52. At the IP interface, the application can use a third-party multimedia protocol stack such as H.323 and MGCP to control calls.

## Available channels

Fusion provides a set of predefined channel types. These channels are created independently of MSPP endpoints, and are used to create a connection between two endpoints. Available MSPP channel types include:

Channel type	Description
G711 full duplex	Processes a full duplex data stream by encoding and decoding voice data according to the G.711 algorithm.
G711 simplex encoder	Processes a simplex data stream by encoding voice data according to the G.711 algorithm.
G711 simplex decoder	Processes a simplex data stream by decoding voice data according to the G.711 algorithm.
G723.1/A full duplex	Processes a full duplex data stream by encoding and decoding voice data according to the G.723.1/A algorithm.
G723.1/A simplex encoder	Processes a simplex data stream by encoding voice data according to the G.723.1/A algorithm.
G723.1/A simplex decoder	Processes a simplex data stream by decoding voice data according to the G.723.1/A algorithm.
G726 full duplex	Processes a full duplex data stream by encoding and decoding voice data according to the G.726 algorithm.
G726 simplex encoder	Processes a simplex data stream by encoding voice data according to the G.726 algorithm.
G726 simplex decoder	Processes a simplex data stream by decoding voice data according to the G.726 algorithm.
G729A/B full duplex	Processes a full duplex data stream by encoding and decoding voice data according to the G.729A/B algorithm.
G729A/B simplex encoder	Processes a simplex data stream by encoding voice data according to the G.729A/B algorithm.
G729A/B simplex decoder	Processes a simplex data stream by decoding voice data according to the G.729A/B algorithm.

Channel type	Description
AMR full duplex	Processes a full duplex data stream by encoding and decoding voice data according to the AMR algorithm.
AMR simplex encoder	Processes a simplex data stream by encoding voice data according to the AMR algorithm.
AMR simplex decoder	Processes a simplex data stream by decoding voice data according to the AMR algorithm.
EVRC full duplex	Processes a full duplex data stream by encoding and decoding voice data according to the EVRC algorithm.
EVRC simplex encoder	Processes a simplex data stream by encoding voice data according to the EVRC algorithm.
EVRC simplex decoder	Processes a simplex data stream by decoding voice data according to the EVRC algorithm.
GSM_FR full duplex	Processes a full duplex data stream by encoding and decoding voice data according to the GSM_FR algorithm.
GSM_FR simplex encoder	Processes a simplex data stream by encoding voice data according to the GSM_FR algorithm.
GSM_FR simplex decoder	Processes a simplex data stream by decoding voice data according to the GSM_FR algorithm.
ILBC_20 full duplex	Processes a full duplex data stream by encoding and decoding voice data according to the ILBC_20 algorithm.
ILBC_20 simplex encoder	Processes a simplex data stream by encoding voice data according to the ILBC_20 algorithm.
ILBC_20 simplex decoder	Processes a simplex data stream by decoding voice data according to the ILBC_20 algorithm.
ILBC_30 full duplex	Processes a full duplex data stream by encoding and decoding voice data according to the ILBC_30 algorithm.
ILBC_30 simplex encoder	Processes a simplex data stream by encoding voice data according to the ILBC_30 algorithm.
ILBC_30 simplex decoder	Processes a simplex data stream by decoding voice data according to the ILBC_30 algorithm.
RTP switching	Provides a mechanism for transferring voice data from one RTP endpoint to another RTP endpoint or from one RTP endpoint to multiple RTP endpoints. Endpoints may be RTP IPv4, RTP IPv6, or any combination of these types.
T.38 fax full duplex	Processes a full duplex data stream by encoding, decoding, modulating, and demodulating T.38 fax data.

**Note:** G.723.1 and G.729A vocoder software is not included with NaturalAccess but can be obtained separately from Dialogic through a license agreement.

After creating and connecting channels, applications can use MSPP functions to query and command these channels. For more information, refer to *Sending MSPP queries and commands* on page 45.

## Available endpoints

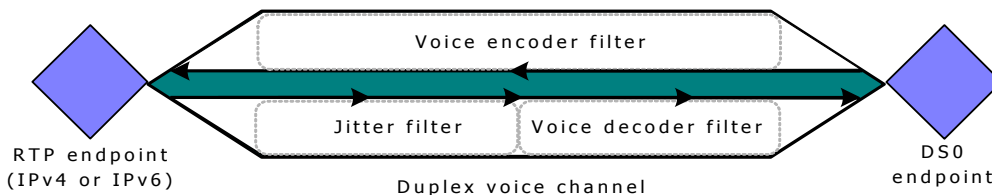
The Fusion MSPP API provides several types of endpoints. Available MSPP endpoints include:

Endpoint type	Description
DS0 full duplex	Entry and exit point for voice or fax over PSTN data.
RTP IPv4 full duplex	Entry and exit point for duplex voice over IP data (includes RTP header within UDP packets) in accordance with IPv4.
RTP IPv4 simplex receive	Entry point for simplex voice over IP data (includes RTP header within UDP packets) in accordance with IPv4.
RTP IPv4 simplex send	Exit point for simplex voice over IP data (includes RTP header within UDP packets) in accordance with IPv4.
RTP IPv6 full duplex	Entry and exit point for duplex voice over IP data (includes RTP header within UDP packets) in accordance with IPv6.
RTP IPv6 simplex receive	Entry point for simplex voice over IP data (includes RTP header within UDP packets) in accordance with IPv6.
RTP IPv6 simplex send	Exit point for simplex voice over IP data (includes RTP header within UDP packets) in accordance with IPv6.
T.38 fax	Entry and exit point for duplex T.38 fax data.
ThroughPacket (duplex)	Entry or exit point for directing multiple voice streams that are multiplexed according to the ThroughPacket algorithm for packet network transmission.

After creating endpoints, applications can use the MSPP functions to query and command the endpoints.

## MSPP filters

Each MSPP connection consists of a linked set of MSPP filters. An MSPP filter is a task or process that performs an operation or set of operations with data that flows through it. All MSPP endpoints and channels are made up of one or more filters that perform specific tasks with the data they receive. Cumulatively, these filters carry out all the tasks performed by the endpoints and channels that make up the connection. The following illustration shows the channels that make up a typical voice channel:



MSPP endpoints and channels use filters in the following way:

- Endpoints are made up of a single filter that translates data between a network specific transport format (for example, DS0 or IP format) and a media-specific format (for example, T.38 fax, G.711 voice).
- Channels consist of one or more MSPP filters that perform specific tasks with data as it moves from one endpoint to another.

Applications can use MSPP API functions to send commands and queries to the filters that make up standard MSPP API components (endpoints and channels). For more information about MSPP channel and endpoint filters, refer to the *Dialogic® NaturalAccess™ Media Stream Protocol Processing API Developer's Manual*.



---

# 4

## Getting started with Fusion

---

### Before using Fusion

---

Before using Fusion software, perform the following steps:

Step	Action	For information, refer to the ...
1	Install the CG board, if any.	CG board manual.
2	Install NaturalAccess software.	<i>NaturalAccess</i> installation booklet.
3	Install the Fusion VoIP API software.	<i>Fusion</i> installation booklet.

**Note:** You must purchase a separate license agreement for certain vocoders (for example, G.723, G.729, or AMR). For more information, contact a Dialogic Services and Support representative.

After you complete these steps, you can configure the system and run demonstration programs to verify that the board and software are installed correctly.

### Getting started tasks

---

This section describes how to perform the following steps:

Step	Action	For more information, refer to...
1	Configure the CG board and boot the board as a Fusion board using <i>oamsys</i> .	<i>Configuring CG boards</i> on page 26, <i>Customizing the board configuration</i> on page 31, or the board's hardware documentation.
2	Run the <i>msppxsr</i> demonstration program to test the MSPP API and CG board DSP resources by creating MSPP endpoints, channels, and connections. Running <i>msppxsr</i> requires two Fusion systems connected to the same network and a device for generating calls at both ends.	<i>MSPP exerciser: msppxsr</i> on page 137.
3	Run CG board utilities to set up the Ethernet interface and gather statistics about transferred data.	<i>Using CG board utilities</i> on page 28 or the appropriate CG board manual.
4	Run the <i>msppsamp</i> demonstration program to transfer data between two Fusion systems.	<i>Fusion nailed-up sample: msppsamp</i> on page 141.

## Configuring CG boards

When you install and boot the CG board with the configuration files provided with NaturalAccess, it is ready for general purpose use. To configure the board to work with Fusion demonstration programs, reboot the board with *oamsys* using the *fusion\_oamsys.cfg* sample system configuration file, and a Fusion sample board keyword file.

Sample board keyword files are located in `\nms\cg\cfg\` (Windows) or `/opt/nms/cg/cfg` (Unix/Linux). The following table is a partial listing of available board keyword files:

Board keyword file	Description
<i>cg6060fusion.cfg</i>	Fusion configuration file for 120 ports on CG 6060 and CG 6060C boards that use E1 interfaces and the on-board IPv4 stack. Also contains commented out configuration for T1 and IPV6.
<i>cg6565fusion.cfg</i>	Fusion configuration file for 240 ports on CG 6565, CG 6565C, and CG 6565E boards that use E1 interfaces and the on-board IPv4 stack. Also contains commented out configuration for T1 and IPV6.

The sample configuration files specify the following CG board settings:

- PSTN (T1/E1) interface
- Ethernet interface
- Board switching mode (for example, standalone or other)
- On-board resource management

Edit the appropriate file so the following PSTN and Ethernet interface configuration information is appropriate for the system. The modified configuration file should include the following system-specific entries:

Configuration	Example
Line interfaces (T1/E1)	<p>The following keywords configure the CG board T1/E1 trunks in the <i>cg6kfusion-t.cfg</i>. In this example, the interfaces are configured as T1 trunks:</p> <pre> NetworkInterface.T1E1[0..3].Type = T1 NetworkInterface.T1E1[0..3].Impedance = DSX1 NetworkInterface.T1E1[0..3].LineCode = B8ZS NetworkInterface.T1E1[0..3].FrameType = ESF NetworkInterface.T1E1[0..3].SignalingType = CAS  DSPStream.VoiceIdleCode[0..3] = 0x7f DSPStream.SignalIdleCode[0..3] = 0x00 </pre>
IPv4 Ethernet interfaces	<p>The following keyword entries define IPv4 Ethernet interfaces for the CG board:</p> <pre> IPC.AddRoute[0].DestinationAddress = 10.102.1.12 IPC.AddRoute[0].Mask = 255.255.0.0 IPC.AddRoute[0].Interface = 1 </pre> <p>The following keyword entries define the default route table entry:</p> <pre> IPC.AddRoute[1].DestinationAddress = 0.0.0.0 IPC.AddRoute[1].Mask = 0.0.0.0 IPC.AddRoute[1].GatewayAddress = 10.102.1.10 </pre> <p>You can also set Ethernet interface configuration information with the <i>cgroute</i> utility. For information about <i>cgroute</i> and board keywords used to configure IPv4 Ethernet interfaces, refer to the CG board documentation.</p> <p>For more information about board keywords used to configure IPv6 Ethernet</p>

Configuration	Example
	interfaces, refer to <i>Configuring CG board Ethernet interfaces</i> on page 36 or to the CG board manual.
IPv6 Ethernet interfaces	<p>The following keyword entries define IPv6 Ethernet interfaces for the CG board:</p> <pre>IPv6.Link[0].Enable = YES IPv6.Link[0].IPSec  = NO IPv6.Link[0].MTU    = 1500 IPv6.Link[0].HopLimit = 64 IPv6.Link[0].EnablePing = YES IPv6.Link[0].ICMPRateLimit = 100 IPv6.Link[0].NDAttempts = 3 IPv6.Link[0].NDRetransTimer = 1000 IPv6.Link[0].NDRachabilityTimer = 30000  IPv6.Link[1].Enable = YES IPv6.Link[1].IPSec  = NO IPv6.Link[1].MTU    = 1500 IPv6.Link[1].HopLimit = 128 IPv6.Link[1].EnablePing = YES IPv6.Link[1].ICMPRateLimit = 100 IPv6.Link[1].NDAttempts = 3 IPv6.Link[1].NDRetransTimer = 1000 IPv6.Link[1].NDRachabilityTimer = 30000</pre> <p>For more information about board keywords used to configure IPv6 Ethernet interfaces, refer to <i>Configuring CG board Ethernet interfaces</i> on page 36 or to the CG board manual.</p>
Board clocking	<p>The following example keywords set the board in standalone mode and to reference its internal clocking from a digital trunk interface (in this case, trunk 1):</p> <pre>Clocking.HBus.ClockMode = STANDALONE Clocking.HBus.ClockSource = NETWORK Clocking.HBus.ClockSourceNetwork = 1</pre> <p>In multiple board systems, you can configure each board to reference its clocking from a primary clock master, and optionally, a secondary clock master.</p>

## Booting CG boards

To boot a CG board:

Step	Action								
1	Edit one of the sample Fusion CG board keyword files to specify settings appropriate for the system.								
2	<p>Edit the <i>fusion_oamsys.cfg</i> file by specifying the following:</p> <table border="1"> <thead> <tr> <th>Entry</th><th>Description</th></tr> </thead> <tbody> <tr> <td>Bus</td><td>Bus number of the CG board.</td></tr> <tr> <td>Slot</td><td>Slot number of the CG board</td></tr> <tr> <td>File</td><td>File name of the CG board keyword file (for example, <i>cg6kfusion-t.cfg</i> or <i>cg6kfusion-e.cfg</i>).</td></tr> </tbody> </table> <p>You can obtain the CG board bus and slot number by running the <i>pciscan</i> utility.</p>	Entry	Description	Bus	Bus number of the CG board.	Slot	Slot number of the CG board	File	File name of the CG board keyword file (for example, <i>cg6kfusion-t.cfg</i> or <i>cg6kfusion-e.cfg</i> ).
Entry	Description								
Bus	Bus number of the CG board.								
Slot	Slot number of the CG board								
File	File name of the CG board keyword file (for example, <i>cg6kfusion-t.cfg</i> or <i>cg6kfusion-e.cfg</i> ).								
3	<p>Enter the following at the command prompt:</p> <pre>oamsys -f</pre>								

For more information about using OAM board configuration tools and *pciscan*, refer to the *Dialogic® NaturalAccess™ OAM System Developer's Manual*.

## Verifying the installation

---

Once you have installed the Fusion software, verify that the MSPP API can use CG board resources by running the *msppxsr* demonstration program.

The *msppxsr* program:

- Creates MSPP endpoints and channels.
- Connects MSPP endpoints and channels.
- Enables the MSPP channels and connections.
- Sends commands and queries to MSPP channels (optional).
- Disables, disconnects, and destroys MSPP endpoints and channels.

For detailed information about *msppxsr*, refer to *MSPP exerciser: msppxsr* on page 137.

## Using CG board utilities

---

Use the following utilities to set up the CG board Ethernet interface and gather statistics about data transferred between Fusion systems:

Program	Description
<i>cgroute</i>	Configures the routing table for a CG board.
<i>cg6kcon</i>	Displays information about the data transmission characteristics of the Fusion system.
<i>cgv6if</i>	Adds, prints, and deletes IPv6 addresses for a CG board.
<i>cgsetkey</i>	Adds, updates, dumps, or flushes IPsec security association database entries and security policy database entries on the board.

Refer to the CG board manual for detailed information about CG board utilities.

## Using msppsamp

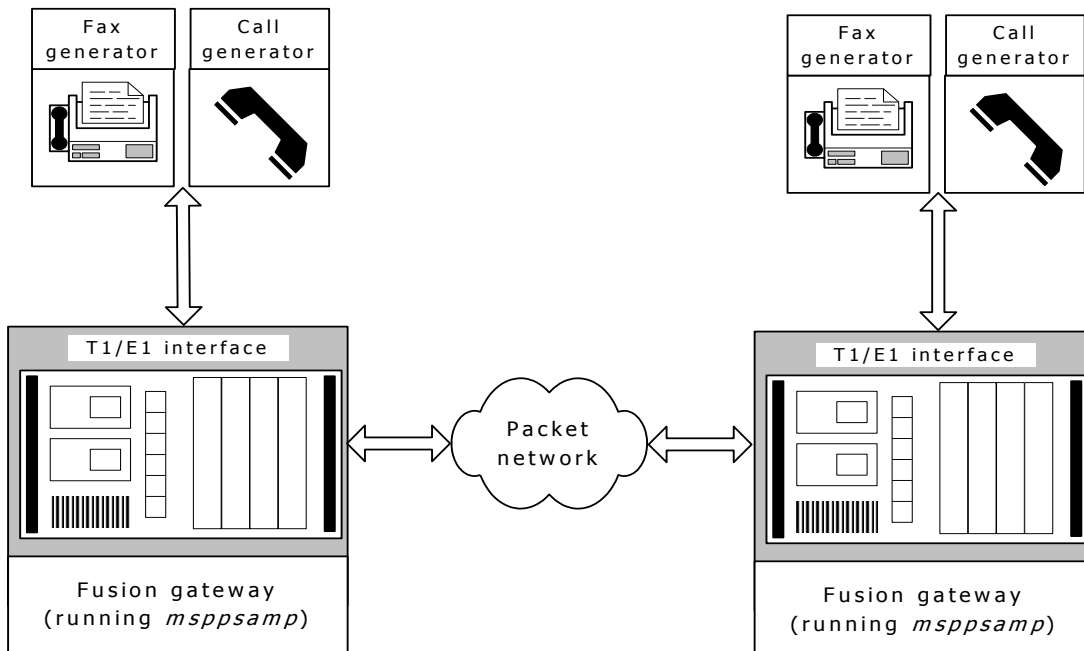
Run *msppsamp* to test data transmission between two Fusion gateway systems. *msppsamp* uses the MSPP service to process and encode/decode voice and fax data while transferring data between PSTN and IP networks.

*msppsamp* performs the following tasks:

- Creates the maximum supported number of MSPP connections between endpoints on the IP network and the PSTN interfaces.
- Responds to incoming calls from the PSTN and enables MSPP connections.
- Provides a mechanism for controlling and querying the filters that make up MSPP connections.
- Initializes and loads the CG board (using *oamsys*) when invoked through the sample command file *demo.cmd*.

### msppsamp illustration

*msppsamp* requires two computers equipped with CG boards and connected over an IP network. Each system must include Fusion software and test equipment to place and receive calls (and, if necessary, transmit and receive fax data). The following illustration shows an overview of an *msppsamp* test environment:



For more information, refer to *Fusion nailed-up sample: msppsamp* on page 141.

## Compiling and linking

When compiling a Fusion application, direct the compiler to the `\nms\include` directory (or `/opt/nms/include` for UNIX) for the following MSPP API header files:

Header file	Description
<i>mspcmd.h</i>	Defines MSPP filter commands and IDs for each filter object.
<i>mspdef.h</i>	Defines functions, parameter structures, events, and errors within the MSPP service.
<i>mspinit.h</i>	Defines initialization parameters for MSPP endpoint and channel filters.
<i>mspobj.h</i>	Defines MSPP objects (such as the filter ID to an ARM filter). It defines unique identifiers that applications can use to communicate with an object. This file identifies the MSPP service objects to which queries and messages are addressed.
<i>mspquery.h</i>	Defines MSPP filter queries for MSPP objects and defines the return data structure for each query.
<i>mspunsol.h</i>	Defines MSPP event IDs and definitions for buffers appended in the unsolicited events.

Applications that use the MSPP API must also link to *mspapi.lib* (or *libmspapi.so* for UNIX) to interface with the MSPP service. They can also link to libraries installed with Fusion software and NaturalAccess. Shared libraries can be found under `\nms\lib` (or `/opt/nms/lib` for UNIX).

Refer to the *Installing Natural Access* for more information about NaturalAccess include and library files. For more information about MSPP filter commands and queries, refer to the *Dialogic® NaturalAccess™ Media Stream Protocol Processing API Developer's Manual*.

For example code from a sample MSPP API application, refer to the *msppsamp* demonstration program source code and to its make file. For information, refer to *Fusion nailed-up sample: msppsamp* on page 141.

---

# 5

## Customizing the Fusion configuration

---

### Customizing the board configuration

---

When you install CG boards, you use the OAM API to specify configuration information for the system and for individual boards within the system. OAM includes the *oamsys* utility, which transfers software modules specified in configuration files to the boards in the system. Use *oamsys* to initialize the boards based on the information in a system configuration file and one or more board keyword files.

Sample configuration files are shipped with Fusion software. These include a sample system configuration file (*fusion\_oamsys.cfg*), and several sample board keyword files described in *Configuring CG boards* on page 26. Use these configuration files with *oamsys* to set up the system. However, before running *oamsys*, you can customize these configuration files according to the characteristics of the system.

For more information about setting up Fusion systems, refer to *Before using Fusion* on page 25. For more information about OAM, refer to the *Dialogic® NaturalAccess™ OAM System Developer's Manual*. Refer to the CG board manual for more information about CG board keyword files.

### Configuring CG board resources

---

The CG boards perform resource management for functions that run on the board's digital signal processors (DSPs). Depending on the programs loaded to them, the DSPs execute functions for encoding, decoding, detecting, and generating voice and telephony signals.

On-board resource management is usually configured to operate on a port as the fundamental unit managed. For PSTN-based applications, a port is equivalent to a circuit-switched call. On-board resource management reserves all of the resources required for a port before the port is used. The resources that are reserved for a port are specified in the Resource[*x*].Definitions keyword and the Resource[*x*].TCPs keyword in the CG board keyword file.

This section assumes that the DSP functions required for a port are loaded on every DSP, with the exception of the signaling DSP (by default DSP 0), which is dedicated to signaling functions. For information about resource keywords and configuration files, refer to the CG board manual.

### DSPs

---

DSP executable code is distributed in data processing module (DPM) files with an *.f54* file extension. Each DPM file contains executable code for a family of algorithms or functions. Each algorithm in that family is known as a data processing function (DPF), and is referenced by a unique ID string. For example, for an echo canceller that has a filter length of 20 milliseconds and an adapt rate of 100 percent, the function ID is *f\_echo\_v3.ln20\_ap100*.

You can also refer to DPFs by specifying hexadecimal numbers generated by combining the family ID number associated with the DPM with the algorithm ID number associated with the DPF. For example, the file *f\_echo\_v3.f54* is the DPM for echo cancellation. All of the DPFs in *f\_echo\_v3.f54* provide a type of echo cancellation functionality. The echo canceller family ID is 0x160000. For an echo canceller that has a filter length of 20 ms. and an adapt rate of 100 percent, the function ID is 0x0A00. Therefore, this particular echo DPF is identified by the hexadecimal number 0x160A00.

### Using f54info

---

To list all function IDs in a DPM file, run *f54info.exe* from the command line. For example, for echo cancellation, enter the following command to obtain a list of all DPFs in the echo family of DPFs:

```
f54info f_echo_v3
```

For information about *f54info.exe*, refer to the CG board manual.

### Using DSP and Resource keywords

---

Keywords in the CG board keyword file specify which DPM files are loaded to each DSP core on the board. By default, the on-board resource manager manages these DSP resources according to entries specified in a board configuration.

Keywords used in the Fusion sample configuration files for configuring and managing DSP resources include:

- DSP.C5x[**x**].Files
- Resource[**n**].Name
- Resource[**n**].Size
- Resource[**n**].TCPs
- Resource[**n**].Definitions
- Resource[**n**].Dsps

Where **x** is the index of a DSP core and **n** is the index of a resource pool.

This topic describes:

- Resource keywords
- Media and call progress masks



## Resource keywords

The following Resource keywords control the way resources are managed for individual CG boards in the system:

Keyword	Description
Resource[ <i>n</i> ].Name	Name for the resource object or pool defined for this board.
Resource[ <i>n</i> ].Size	Number of resource definition objects (for example, channels or ports) managed by the on-board resource manager.
Resource[ <i>n</i> ].TCPs	Trunk control programs (TCPs) used for setting up and tearing down calls. These TCPs must be loaded to the board through the TCPFiles[ <i>x</i> ] keyword in the board keyword file.
Resource[ <i>n</i> ].Definitions	List of DPFs that are available for each port and determines when functions execute in relation to each other.
Resource[ <i>n</i> ].Dsps	<p>List of DSP cores that are assigned to this resource object (pool). It is possible to define multiple pools (Fusion or otherwise) thus it can be a partial list of DSP cores.</p> <p><b>Notes:</b></p> <ul style="list-style-type: none"> <li>DSPs are numbered starting with DSP 0.</li> <li>Resource pools have a core pair limitation in that DSP 'n' (n=0,2,4,...) and DSP 'n+1' are core pairs (e.g., DSP 0 and DSP 1), and DSP 2 and DSP 3 are each core pairs.</li> <li>If the DSP cores listed do not exist on the board, the DSP cores will be ignored and will not be booted or used.</li> <li>A Fusion resource pool may not share the same core pair with another non-Fusion (e.g., IVR) resource pool.</li> <li>For example, the following configuration is allowed: <ul style="list-style-type: none"> <li>IVR pool: DSPs 8 – 21</li> <li>Fusion pool: DSPs 22 – 45</li> </ul> </li> <li>However, the following configuration is NOT allowed: <ul style="list-style-type: none"> <li>IVR pool: DSPs 8 – 20</li> <li>Fusion pool: DSPs 21 – 45</li> </ul> </li> </ul>

For on-board resource management to work correctly, other keywords in the board keyword file must match what is being defined by the Resource keywords in the file. Only DPFs that are specified in the Resource[*x*].Definitions keyword in the configuration file are recognized and managed by the resource manager. The reverse is also true. That is, if you specify a DPF in the Resource[*x*].Definitions keyword in the configuration file that is not in the DSP.C5X[*x*].File list of DPMs loaded to the DSPs, the on-board resource manager returns an error when reserving resources.

The following resources are defined in the *cg6kfusion-t.cfg* file. This default resource definition string is common to all Fusion sample board keyword files:

```
Resource[0].Name      = RSC1
Resource[0].Size      = 120
Resource[0].TCPs      = WNK0
Resource[0].Definitions = ((dtmf.det_all & f_echo_v3.ln20_ap25) & \
    ((f_g711.cod & f_g711.dec) | \
    (f_g723.cod & f_g723.dec) | \
    (f_g729a.cod & f_g729a.dec) | \
    (f_g726.cod & f_g726.dec) | \
    (f_faxt38.relay)))
Resource[0].Dsps = 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25 26 27 28 29 30 31
```

In this example, the board configuration specifies that the board uses the wink start protocol (wnk0) to set up and tear down a maximum of 120 voice or fax channels. This configuration also specifies that the following DPFs are conditionally available for each port:

DPF ID	Description
dtmf.det_all	DTMF detector with silence, clear-down, and CED tone detection
f_echo_v3.ln20_ap25	Echo canceller with a filter length of 20 ms. and an adapt rate of 25%
f_g711.cod	G.711 encoder
f_g711.dec	G.711 decoder
f_g711vad.cod	G.711 VAD encoder
f_g711vad.dec	G.711 VAD decoder
f_gsm_fr.cod	GSM full rate encoder
f_gsm_fr.dec	GSM full rate decoder
f_g723.cod	G.723.1/A encoder
f_g723.dec	G.723.1/A decoder
f_g729a.cod	G.729A/B encoder
f_g729a.dec	G.729A/B decoder
f_g726.cod	G.726 encoder
f_g726.dec	G.726 decoder
f_faxt38.relay	T.38 fax relay
f_g728.cod	G.728 encoder
f_g728.dec	G.728 decoder
f_amr.cod	AMR encoder
f_amr.dec	AMR decoder
f_evrc.cod	EVRC encoder
f_evrc.dec	EVRC decoder
f_ilbc_20.cod	ILBC 20 ms encoder
f_ilbc_20.dec	ILBC 20 ms decoder
f_ilbc_30.cod	ILBC 30 ms encoder
f_ilbc_30.dec	ILBC 30 ms decoder

The AND (&) operator indicates that the two DPFs on either side of this character executes at the same time. The OR operator (|) indicates that DPFs on either side of this character are mutually exclusive. Therefore, the example `Resource[x].Definitions` statement indicates that the dtmf and echo DPFs execute simultaneously with *one* of the following DPFs:

- G.711 encoding and decoding (DPFs `f_g711.cod` and `f_g711.dec`).
- G.723.1/A encoding and decoding (DPFs `f_g723.cod` and `f_g723.dec`).
- G.726 encoding and decoding (DPFs `f_g726.cod` and `f_g726.dec`).
- G.729A/B encoding and decoding (DPFs `f_g729a.cod` and `f_g729a.dec`).
- T.38 fax relay encoding and decoding (DPF `f_faxt38.relay`).

Other CG board Resource keywords include:

- `Resource[x].Name`
- `Resource[x].Size`
- `Resource[x].TCPs`

For more information about Resource board keywords, refer to the CG board manual.

### Media and call progress masks

---

Natural Access uses media masks and call progress masks to control on-board resource management. These masks indicate the DPFs that execute on the board during particular states of a call control protocol (TCP). If any of the bits in either of these two masks are set, the DPFs associated with the set bits must be specified in the `Resource[x].Definitions` keyword. Since the bits in the media mask are set by default, the DPFs that correspond to these set bits are specified in the default `Resource[x].Definitions` keyword in the CG board keyword file. For more information about NaturalAccess media masks and call progress masks, refer to the *Dialogic® NaturalAccess™ Software Developer's Manual*.

## Configuring CG board Ethernet interfaces

The CG boards provide two Ethernet connections that can be configured to operate in IPv4 stack mode, IPv6 stack mode, or dual stack mode. There are several methods you can use to set the board's IP addresses and configure its Ethernet interfaces. The method you use depends on the application's requirements and the way you use the board.

You can configure the CG board's Ethernet interface through keywords in the board keyword file using the *oamsys* utility. Every time you use *oamsys* to re-configure the board, the IP address information is re-configured for the board.

Other methods for configuring the Ethernet interface include:

- Using the *cgroute* utility. For information, refer to the CG board manual.
- Using the OAM service API to configure the board from an application. For information, refer to the *Dialogic® NaturalAccess™ OAM API Developer's Manual*.

This topic describes:

- Setting Ethernet interfaces to use the IPv4 stack
- Setting Ethernet interfaces to use the IPv6 stack
- Setting up the board to run in dual IPv4 and IPv6 stack mode

For more information about configuring the CG board Ethernet interfaces, refer to the appropriate CG board manual.

### Setting Ethernet interfaces to use the IPv4 stack

When configured in IPv4 stack mode, the IPv4 addressing and gateway configuration information for each CG board resides in a separate board keyword file. Each time you use *oamsys* to re-configure the board, the IP address information is re-configured for the board. In addition, if the application uses RTP IPv4 endpoints, the associated CG board Ethernet interfaces must be configured in either IPv4 stack mode or dual IPv4 and IPv6 stack mode.

The following CG board keywords configure CG board Ethernet interfaces in IPv4 stack mode:

Keyword	Description
IPC.AddRoute[x].DestinationAddress	Specifies the IPv4 address of a CG board Ethernet interface. There are 30 routes available in dual subnet mode and 15 routes available in redundancy mode.
IPC.AddRoute[x].GatewayAddress	Specifies the IPv4 address of the network router.
IPC.AddRoute[x].Interface	Specifies the number (1 or 2) of the Ethernet interface you are configuring.
IPC.AddRoute[x].Mask	Specifies the subnet mask for the IPv4 address specified in IPC.AddRoute[x].DestinationAddress.

The following example shows how to use `IPC.AddRoute` statements in a sample CG board keyword file to specify the board's IPv4 address, subnet mask, and gateway IPv4 address:

```
IPC.AddRoute[0].DestinationAddress = 10.102.64.151
IPC.AddRoute[0].Mask = 255.255.191.0
IPC.AddRoute[0].Interface = 1

#Gateway IPv4 Address, subnet mask, and gateway IP address.
IPC.AddRoute[1].DestinationAddress = 0.0.0.0
IPC.AddRoute[1].Mask = 0.0.0.0
IPC.AddRoute[1].GatewayAddress = 10.102.64.10
```

In this example, the first three IPC entries specify the IPv4 address and mask of a CG board. The second three entries configure the address of the gateway.

### Setting Ethernet interfaces to use the IPv6 stack

When the CG board is configured in IPv6 stack mode, Ethernet interfaces can use features such as address autoconfiguration neighbor discovery, and IP security.

The following CG board keywords are used to configure the CG board Ethernet interfaces in IPv6 stack mode:

Keyword	Description
IPv6.Link[ <b>x</b> ].Enable	Enables or disables IPv6 on the specified Ethernet interface.
IPv6.Link[ <b>x</b> ].IPSec	Enables or disables IPv6 security (IPSec) on the specified Ethernet interface.
IPv6.Link[ <b>x</b> ].MTU	Specifies the IPv6 maximum transmission unit (MTU) for the Ethernet interface.
IPv6.Link[ <b>x</b> ].HopLimit	Specifies the default IPv6 hop limit value (that is, the maximum number of routers through which a datagram travels) for the Ethernet interface.
IPv6.Link[ <b>x</b> ].EnablePing	Enables or disables IPv6 PING on the specified Ethernet interface.
IPv6.Link[ <b>x</b> ].ICMPRateLimit	Specifies the IPv6 ICMP rate limit (that is, the maximum amount of ICMP error messages per second that can be sent) for the Ethernet interface.
IPv6.Link[ <b>x</b> ].NDAAttempts	Specifies the neighbor discovery attempt (NDA) limit for the Ethernet interface.
IPv6.Link[ <b>x</b> ].NDRetransTimer	Specifies in milliseconds, the neighbor discovery re-transmission timer for the Ethernet interface.
IPv6.Link[ <b>x</b> ].NDRachabilityTimer	Specifies, in milliseconds, the neighbor discovery reachability timer duration for the Ethernet interface.

If the application uses RTP IPv6 endpoints, the associated CG board Ethernet interfaces must be configured either in IPv6 stack mode or in dual IPv4 and IPv6 stack mode.

The following sample shows the segment of a CG board keyword file that specifies the board's Ethernet interfaces in IPv6 mode:

```
IPv6.Link[0].Enable = YES
IPv6.Link[0].IPSec  = NO
IPv6.Link[0].MTU    = 1500
IPv6.Link[0].HopLimit = 64
IPv6.Link[0].EnablePing = YES
IPv6.Link[0].ICMPRateLimit = 100
IPv6.Link[0].NDAttempts = 3
IPv6.Link[0].NDRetranTimer = 1000
IPv6.Link[0].NDRachabilityTimer = 30000

IPv6.Link[1].Enable = YES
IPv6.Link[1].IPSec  = NO
IPv6.Link[1].MTU    = 1500
IPv6.Link[1].HopLimit = 128
IPv6.Link[1].EnablePing = YES
IPv6.Link[1].ICMPRateLimit = 100
IPv6.Link[1].NDAttempts = 3
IPv6.Link[1].NDRetranTimer = 1000
IPv6.Link[1].NDRachabilityTimer = 30000
```

### Setting up the board to run in dual IPv4 and IPv6 stack mode

CG boards can implement a dual IPv4 and IPv6 stack that allows applications to configure the board in any of the following modes:

IP stack mode	Description
IPv4 only	CG board default mode. You configure specific IPv4 addresses and routing information with the IPC.AddRoute keywords.
IPv6 only	Enable by setting the IPv6.Link[ <i>x</i> ].Enable keyword to YES for a particular Ethernet interface. The IPv4 stack remains passive if no IPv4 addresses are configured with IPC.AddRoute keywords.
Dual IPv4 and IPv6 stack	Add IPv4 addresses with IPC.AddRoute keywords, and enable IPv6 capability on a particular interface with the IPv6.Link[ <i>x</i> ].Enable keyword.

When CG board Ethernet interfaces are configured in dual IPv4 and IPv6 stack mode, applications can create both IPv4 and IPv6 endpoints associated with the interfaces. When IPv6 is enabled on the second port, IPv4 redundancy is disabled.

You can configure CG boards Ethernet interfaces in dual IPv4 and IPv6 stack mode in one of the following ways:

- Configure the Ethernet interface separately for IPv4 and IPv6 support.
- Configure separate protocols for separate Ethernet links.
- Configure both protocols for either or both links.

For more information about configuring CG boards in dual IPv4 and IPv6 mode, refer to the CG board manual.

## Sample E1 system configuration file

The following sample configuration file *cg6565fusion.cfg* is provided with the Fusion software. This configuration file configures a typical CG board to boot up in standalone mode so that it runs 240 ports:

```
#
#   cg6565fusion.cfg
#   CG 6565 configuration file
#
#   This file configures the board to run Fusion with NOCC in E1
#
#-----
#   IP V4 Address, subnet mask, and gateway IP address
#-----
# Note: the IP configuration below is for a Ethernet Failover
# THIS CONFIGURATION FILE WILL FAIL UNLESS THE VARIABLE STRINGS
# BELOW ARE REPLACE WITH REAL IP ADDRESSES.
IPC.AddRoute[0].DestinationAddress = x.x.x.x
IPC.AddRoute[0].Mask = y.y.y.y
IPC.AddRoute[0].Interface = 1
#IPC.AddRoute[1].DestinationAddress = 0.0.0.0
#IPC.AddRoute[1].Mask = 0.0.0.0
#IPC.AddRoute[1].GatewayAddress = z.z.z.z
#####
#-----
#   Uncomment this section to use IPV6
#-----
#   The following IPv6.Link.XXX enable both ethernet interfaces
#   for IPv6 with default values - refer to the CG 6565 board
#   hardware manual for more details.
#####
#IPv6.Link[0].Enable = YES
#IPv6.Link[0].IPSec = NO
#IPv6.Link[0].MTU = 1500
#IPv6.Link[0].HopLimit = 64
#IPv6.Link[0].EnablePing = YES
#IPv6.Link[0].ICMPRateLimit = 100
#IPv6.Link[0].NDAttempts = 3
#IPv6.Link[0].NDRetranTimer = 1000
#IPv6.Link[0].NDReachabilityTimer = 30000
#
#IPv6.Link[1].Enable = YES
#IPv6.Link[1].IPSec = NO
#IPv6.Link[1].MTU = 1500
#IPv6.Link[1].HopLimit = 128
#IPv6.Link[1].EnablePing = YES
#IPv6.Link[1].ICMPRateLimit = 100
#IPv6.Link[1].NDAttempts = 3
#IPv6.Link[1].NDRetranTimer = 1000
#IPv6.Link[1].NDReachabilityTimer = 30000
#####
# Throughpacket configuration settings
#
#Configures the Throughpacket DLM at boot time.
#TPKT.Enable = 1
#
# UDP port numbers for Throughpacket Simple protocol.
#TPKT.SimpleRxPort = 49152
#TPKT.SimpleTxPort = 49152
#
# UDP port numbers for Throughpacket complex protocol.
#TPKT.ComplexRxPort = 49153
#TPKT.ComplexTxPort = 49153
#
# Number of conditions specified for Throughpacket data transmission.
#TPKT.NumberOfComplexForwardConditions = 4
#
# Total conditions possible. This value MUST always be set to 8.
#TPKT.ComplexForward.Count = 8
```

```

#
# Data transmission conditions for Throughpacket complex protocol
#TPKT.ComplexForward[0].LifeTimeTicks = 0
#TPKT.ComplexForward[0].DestinationPacketSize = 1440
#TPKT.ComplexForward[1].LifeTimeTicks = 1
#TPKT.ComplexForward[1].DestinationPacketSize = 980
#TPKT.ComplexForward[2].LifeTimeTicks = 2
#TPKT.ComplexForward[2].DestinationPacketSize = 700
#TPKT.ComplexForward[3].LifeTimeTicks = 3
#TPKT.ComplexForward[3].DestinationPacketSize = 1
#TPKT.ComplexForward[4].LifeTimeTicks = 0
#TPKT.ComplexForward[4].DestinationPacketSize = 0
#TPKT.ComplexForward[5].LifeTimeTicks = 0
#TPKT.ComplexForward[5].DestinationPacketSize = 0
#TPKT.ComplexForward[6].LifeTimeTicks = 0
#TPKT.ComplexForward[6].DestinationPacketSize = 0
#TPKT.ComplexForward[7].LifeTimeTicks = 0
#TPKT.ComplexForward[7].DestinationPacketSize = 0
#####
Clocking.HBus.ClockMode = STANDALONE
Clocking.HBus.ClockSource = OSC
# DSP.C5x[x].Os strongly recommended - adjust the dsp number range accordingly
DSP.C5x[0..95].Os = dspos6u
#-----
# NOTE: T1 configuration
#-----
#NetworkInterface.T1E1[0..15].Type = T1
#NetworkInterface.T1E1[0..15].Impedance = DSX1
#NetworkInterface.T1E1[0..15].LineCode = B8ZS
#NetworkInterface.T1E1[0..15].FrameType = ESF
#NetworkInterface.T1E1[0..15].SignalingType = RAW
#DSPStream.VoiceIdleCode[0..15] = 0x7F
#DSPStream.SignalIdleCode[0..15] = 0x00
#DSP.C5x[0..95].Libs = cg6klibu f_shared
#DSP.C5x[0..95].XLaw = MU_LAW
#-----
# NOTE: E1 configuration
#-----
NetworkInterface.T1E1[0..15].Type = E1
NetworkInterface.T1E1[0..15].Impedance = G703_120_OHM
NetworkInterface.T1E1[0..15].LineCode = HDB3
NetworkInterface.T1E1[0..15].FrameType = CEPT
NetworkInterface.T1E1[0..15].SignalingType = RAW
DSPStream.VoiceIdleCode[0..15] = 0xD5
DSPStream.SignalIdleCode[0..15] = 0x0D
DSP.C5x[0..95].Libs = cg6kliba f_shared
DSP.C5x[0..95].XLaw = A_LAW
#-----
# Hardware Echo Cancellation
# NOTE: it is in by pass by default
# NOTE: uncomment the following two keyword lines to enable and set the XLaw accordingly
#-----
# HardwareEcho.EchoChipEnabled = YES
# HardwareEcho.XLaw = A_LAW
#-----
# Resource management
#-----
# Before modifying this resource definition string refer to the CG6565
# Installation and Developers Manual.
# 1. Load dtmf, f_g711, f_g726, f_faxt38.
# 2. Do not load f_echo
#####
Resource[0].Name = RSC1
Resource[0].Size = 240
Resource[0].TCPs = nocc
Resource[0].StartTimeSlot = 0

Resource[0].Definitions = ((dtmf.det_sil_clrdwn_ced & \
((f_g726.cod_rfc2833 & f_g726.dec_rfc2833) | \
(f_g711.cod_rfc2833 & f_g711.dec_rfc2833) | \
f_faxt38.relay)) | \

```



```

        (dtmf.det_all & \
        ((f_g726.cod & f_g726.dec) | \
        (f_g711.cod & f_g711.dec) | \
        f_faxt38.relay)))

# Load dtmf, f_g711, f_g723, f_faxt38.
#####
# Resource[0].Definitions = ((dtmf.det_sil_clrdwn_ced & \
# ((f_g723.cod_rfc2833 & f_g723.dec_rfc2833) | \
# (f_g711.cod_rfc2833 & f_g711.dec_rfc2833) | \
# f_faxt38.relay)) | \
# (dtmf.det_all & \
# ((f_g723.cod & f_g723.dec) | \
# (f_g711.cod & f_g711.dec) | \
# f_faxt38.relay)))
# Load dtmf, f_g711, f_g729a, f_faxt38.
#####
# Resource[0].Definitions = ((dtmf.det_sil_clrdwn_ced & \
# ((f_g723.cod_rfc2833 & f_g723.dec_rfc2833) | \
# (f_g711.cod_rfc2833 & f_g711.dec_rfc2833) | \
# f_faxt38.relay)) | \
# (dtmf.det_all & \
# ((f_g729a.cod & f_g729a.dec) | \
# (f_g711.cod & f_g711.dec) | \
# f_faxt38.relay)))
# Load dtmf, f_g711, f_amr, f_faxt38.
#####
# Resource[0].Definitions = ((dtmf.det_sil_clrdwn_ced & \
# ((f_amr.cod_rfc2833 & f_amr.dec_rfc2833) | \
# (f_g711.cod_rfc2833 & f_g711.dec_rfc2833) | \
# f_faxt38.relay)) | \
# (dtmf.det_all & \
# ((f_amr.cod & f_amr.dec) | \
# (f_g711.cod & f_g711.dec) | \
# f_faxt38.relay)))
# Load dtmf, f_g711, f_evrc, f_faxt38.
#####
# Resource[0].Definitions = ((dtmf.det_sil_clrdwn_ced & \
# ((f_evrc.cod_rfc2833 & f_evrc.dec_rfc2833) | \
# (f_g711.cod_rfc2833 & f_g711.dec_rfc2833) | \
# f_faxt38.relay)) | \
# (dtmf.det_all & \
# ((f_evrc.cod & f_evrc.dec) | \
# (f_g711.cod & f_g711.dec) | \
# f_faxt38.relay)))
# Load dtmf, f_g711, f_ilbc_20, f_faxt38.
#####
# Resource[0].Definitions = ((dtmf.det_sil_clrdwn_ced & \
# ((f_ilbc_20.cod_rfc2833 & f_ilbc_20.dec_rfc2833) | \
# (f_g711.cod_rfc2833 & f_g711.dec_rfc2833) | \
# f_faxt38.relay)) | \
# (dtmf.det_all & \
# ((f_ilbc_20.cod & f_ilbc_20.dec) | \
# (f_g711.cod & f_g711.dec) | \
# f_faxt38.relay)))
# Load dtmf, f_g711, f_ilbc_30, f_faxt38.
#####
# Resource[0].Definitions = ((dtmf.det_sil_clrdwn_ced & \
# ((f_ilbc_30.cod_rfc2833 & f_ilbc_30.dec_rfc2833) | \
# (f_g711.cod_rfc2833 & f_g711.dec_rfc2833) | \
# f_faxt38.relay)) | \
# (dtmf.det_all & \
# ((f_ilbc_30.cod & f_ilbc_30.dec) | \
# (f_g711.cod & f_g711.dec) | \
# f_faxt38.relay)))
# Load dtmf, f_g711, f_ilbc_30, f_faxt38.
#####
# Resource[0].Definitions = ((dtmf.det_sil_clrdwn_ced & \
# ((f_gsm_fr.cod_rfc2833 & f_gsm_fr.dec_rfc2833) | \
# (f_g711.cod_rfc2833 & f_g711.dec_rfc2833) | \
# f_faxt38.relay)) | \

```

```

#                               (dtmf.det_all & \
#                               ((f_gsm_fr.cod & f_gsm_fr.dec) | \
#                               (f_g711.cod & f_g711.dec) | \
#                               f_faxt38.relay)))

# NOTE: If the DSP cores listed below do not exist on the board, the DSP cores will
# be ignored and will not be booted or used
Resource[0].Dsps = 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 \
                  24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
47 \
                  48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70
71 \
                  72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94
95
DebugMask                               = 0x0
DLMFiles[0]                             = cg6565fusion
#DLMFiles[1]                             = cg6565tpkt

```

---

# 6

## Managing MSPP connections

---

### Creating NaturalAccess contexts and queues

---

Before calling functions from the MSPP API, initialize NaturalAccess, by invoking **ctaInitialize** and specifying the names of the services and service managers that the application will use. For a list of standard NaturalAccess services, refer to the *Dialogic® NaturalAccess™ Software Developer's Manual*.

After initializing Natural Access and its services, create event queues and contexts with the following functions:

Function	Description
<b>ctaCreateQueue</b>	Creates an event queues and specifies the service managers to attach to the queue (for example, the MSPP service uses the MSPMGR).
<b>ctaCreateContext</b>	Creates a context and associated with a queue handle ( <b>ctaqueuehd</b> ) returned from <b>ctaCreateQueue</b> . All events for services on the context are received in the attached event queue.

**ctaCreateContext** returns a context handle (**ctahd**). When an application invokes MSPP functions, it must specify a context handle. Events communicated back to the application are associated with the specified handle. Refer to the *Dialogic® NaturalAccess™ Software Developer's Manual* for more information about NaturalAccess programming models and managing contexts and queues.

### Opening MSPP service instances

---

**ctaOpenServices** opens one or more service instances on a particular context. To open an MSPP service instance on a context, invoke **ctaOpenServices** with the following:

- A **ctahd**.
- Service name (MSP) and associated service manager (MSPMGR).
- The board number, stream and timeslot, and mode to associate with the service instance.

For more information about arguments, errors, and events, refer to the *Dialogic® NaturalAccess™ Software Developer's Manual*.

### Example: Opening the ADI service and MSPP services on a context

In some cases, the application must open multiple services on the contexts associated with MSPP service instances. For example, when an application wants to perform silence detection with the voice data the passes through MSPP connections, it must open an ADI service instance with each context it uses to create DS0 endpoints. To do this, the application must open the MSPP and ADI services on the context and specify (in the CTA\_MVIP\_ADDR structure) following address and mode information for the ADI service:

CTA_MVIP_ADDR parameter	Value	Description
services[0].mvipaddr.board	NA	Name of the CG board on which to open the service.
svclist.mvipaddr.stream	0	Stream associated with the DSP port. Dialogic recommends using stream 0.
svclist.mvipaddr.timeslot	0 - 127	Timeslot associated with the DSP port.
svclist.mvipaddr.mode	ADI_VOICE_DUPLEX	Mode of operation for DSP resources allocated to the service instance.  For more information ADI service instances, refer to the <i>Dialogic® NaturalAccess™ Alliance Device Interface API Developer's Manual</i> .

**Note:** When the application uses **mspCreateEndpoint** to create DS0 endpoints, it must specify the same timeslot in the DS0\_ENDPOINT\_ADDR structure that it specified in the CTA\_MVIP\_ADDR substructure when it opened the MSPP API on the context.

### Setting up MSPP connections

Applications create end-to-end media channels by creating MSPP endpoints and channels and joining them to form simplex or duplex connections. The following table shows the MSPP functions used to set up connections and transfer data across a Fusion gateway:

Function	Description
<b>mspCreateEndpoint</b>	Creates an endpoint and associates it with a context. The type of endpoint and its associated timeslot is provided in an ADDR structure. The timeslot must be the same timeslot used to open the MSPP service instance on the context.
<b>mspCreateChannel</b>	Creates a media channel and associates it with a context. The type of channel is specified in an ADDR structure through an alphanumeric text string.
<b>mspConnect</b>	Connects a media channel to two endpoints.
<b>mspEnableChannel</b>	Enables data to pass between two endpoints through an MSPP channel.
<b>mspEnableEndpoint</b>	Enables data to pass through an endpoint that was previously disabled.

## Tearing down MSPP connections

Use the following functions to tear down MSPP connections:

Function	Description
<b>mspDisableChannel</b>	Interrupts the flow of data on a specified MSPP channel.
<b>mspDisableEndpoint</b>	Interrupts the flow of data through a specified MSPP endpoint.
<b>mspDisconnect</b>	Disassociates the MSPP endpoints and MSPP channel in a specified connection.
<b>mspDestroyEndpoint</b>	Destroys a specified MSPP endpoint and disables and disconnects the associated connection if the endpoint is part of an enabled connection.
<b>mspDestroyChannel</b>	Destroys a specified MSPP channel and disables and disconnects the associated connection if the channel is part of an enabled connection.

When tearing down MSPP connections, applications must invoke MSPP service functions in the same order that they appear in the preceding table, with the exception that **mspDestroyChannel** can precede or follow **mspDestroyEndpoint**.

For more information about MSPP functions, refer to the *Dialogic® NaturalAccess™ Media Stream Protocol Processing API Developer's Manual*.

## Sending MSPP queries and commands

Once the application has initialized the MSPP service, it can send queries to the MSPP service to obtain global, connection, and component specific information. The MSPP service supports the following types of queries and commands:

Query or command	Description
Filter queries and commands	Returns specific information about the component filters that make up MSPP endpoints and channels.
System-level queries	Returns board or session specific information concerning resource usage or data transmission status.

### Filter queries and commands

Once you create MSPP endpoints and connect MSPP channels, you can modify the behavior or configuration of these objects with **mspSendCommand**. When applications implement voice/fax switchover, they use **mspSendCommand** to switch the DS0 endpoint from voice to T.38 fax mode and back.

Applications use **mspSendQuery** to elicit status or configuration information for a specific filter in a connected MSPP channel, and must use **mspReleaseBuffer** to release buffers returned with **mspSendCommand** and **mspSendQuery** events.

MSPP filters (channel and endpoint) need to be in particular states when they receive commands. This varies from filter to filter. For more information about sending MSPP filter commands and queries, refer to the *Dialogic® NaturalAccess™ Media Stream Protocol Processing API Developer's Manual*.

## System-level queries

The MSPP service supports a set of queries associated with board and session-level resources rather than with specific MSPP endpoint filters. These queries allow applications to monitor the status of board-level CPU resources and Ethernet connections while applications are running.

The MSPP service supports the following system queries:

Query	Description
MSP_QRY_ETHERNET_LINK_STATUS	Returns information about the board's Ethernet link connections on a session-by-session basis. This request allows the user to retrieve at any time the physical status of each of the CG board's Ethernet links.
MSP_QRY_CPU_USAGE	Returns information about the board's current and average CPU utilization.

When applications use these queries, they invoke **mspSendQuery** while specifying the query name and any valid MSPP endpoint or channel handle associated with the board that they want to query. Because the queries are board-specific, the identity of the MSPP endpoint or channel is irrelevant. Within the query however, the application must specify the constant `MSP_SYSTEM` as the filter ID since there is no filter associated with these queries.

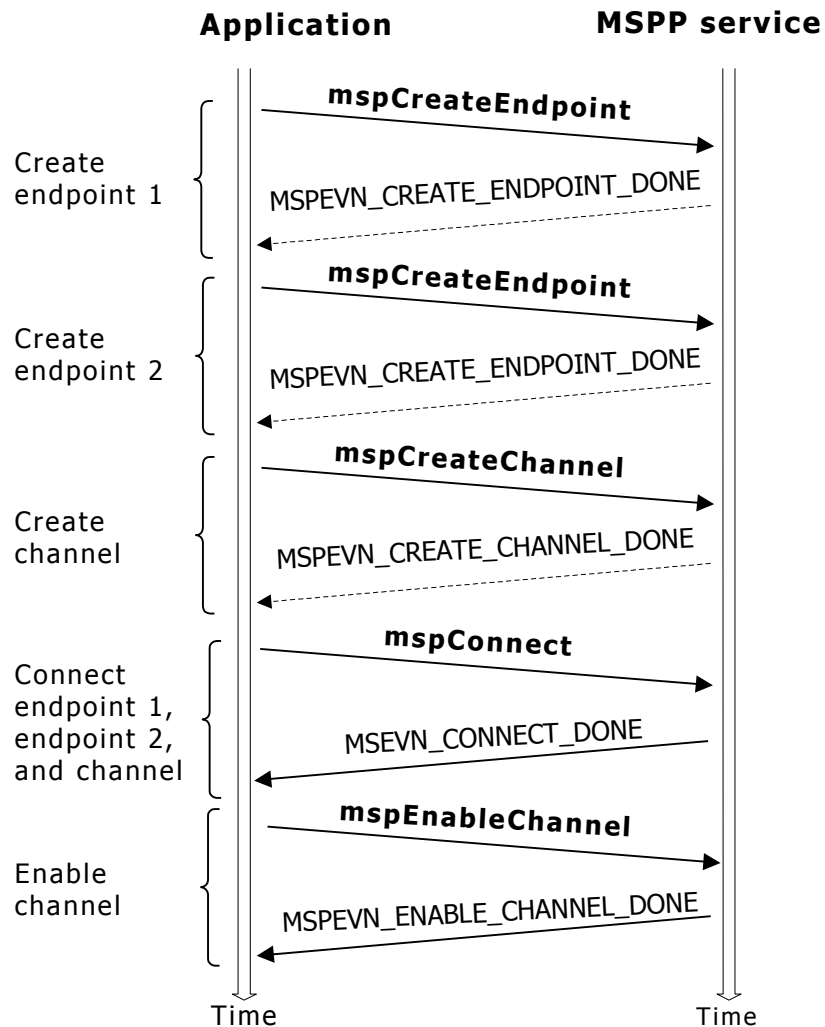
Both queries return buffered data in response to **mspSendQuery**.

`MSP_QRY_ETHERNET_LINK_STATUS` returns an `msp_ETH_STATUS` structure that indicates the status (up or down), mode, and speed of the CG board's Ethernet interface. `MSP_QRY_CPU_USAGE` returns a `CPU_UTIL` structure that indicates the current and average (over the last 16 seconds) CPU utilization on the CG board.

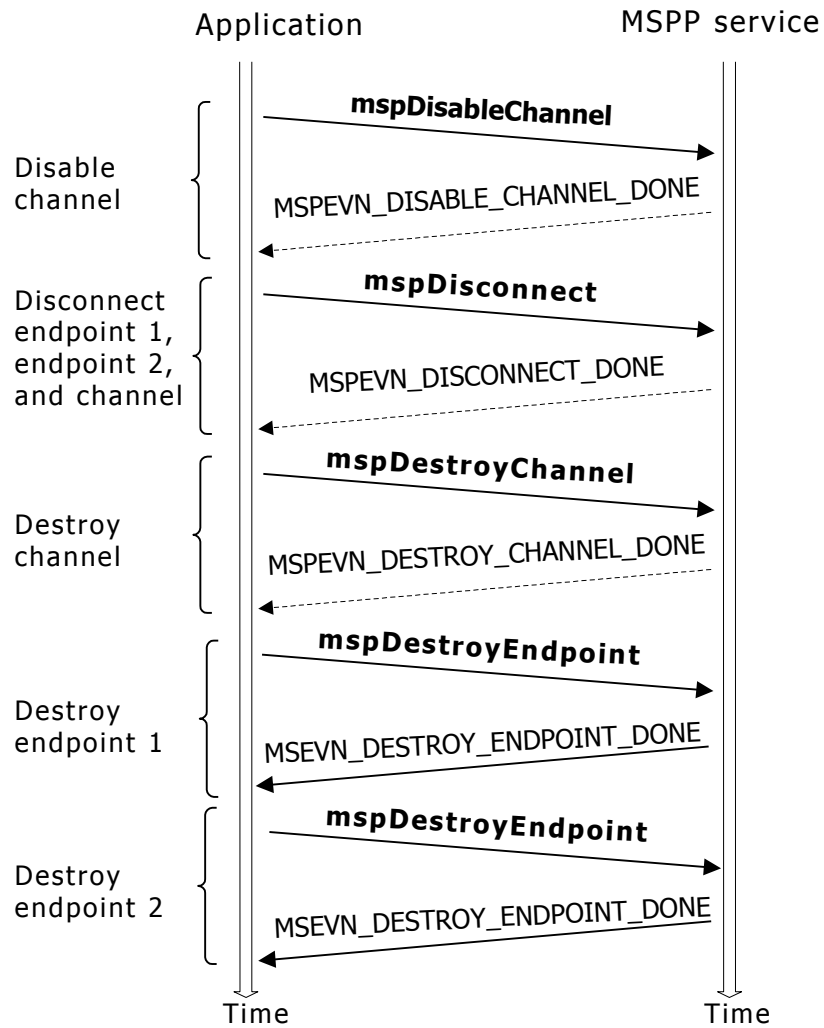
Route availability unsolicited events also provide the application with status information concerning particular RTP or UDP sessions. For more information, refer to *RTP and UDP endpoint route availability events* on page 65.

## MSPP function sequence

The following illustration shows the function sequence for setting up MSPP connections:



The following illustration shows the function sequence for tearing down MSPP connections:



Applications can interrupt a connection either by disabling the channel with **mspDisableChannel**, or by disabling one of its component connection endpoints with **mspDisableEndpoint**.

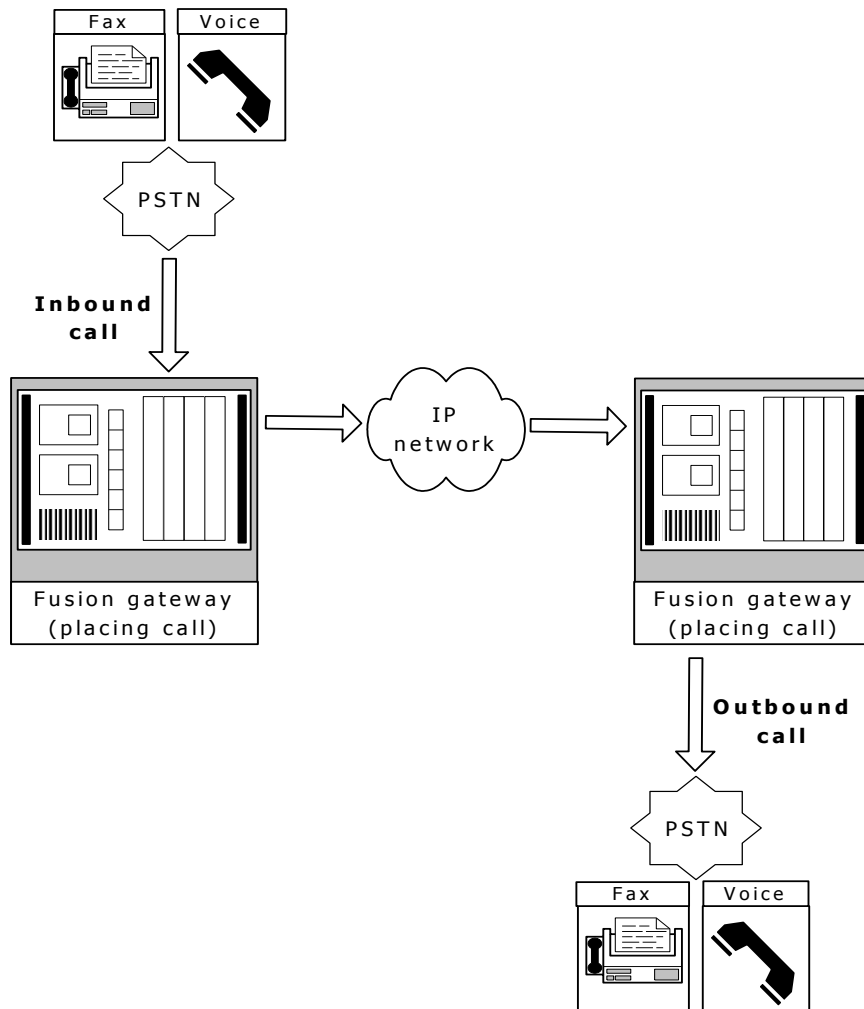


## Connecting MSPP channels during call set up

The NaturalCallControl (NCC) API is an extension to the NaturalAccess environment that provides applications with functions for performing PSTN call control. The NCC programming model provides a model for performing general purpose call control tasks such as receiving inbound calls and placing outbound calls.

Because Fusion gateway applications are connected to an IP network as well as a PSTN network, applications can respond to inbound calls and place outbound calls differently than the typical NCC model. Fusion applications must connect, enable, and disable MSPP connections while controlling inbound and outbound calls.

The following illustration shows two Fusion applications placing and receiving calls to transfer voice or fax data between two PSTN networks through an IP network:



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

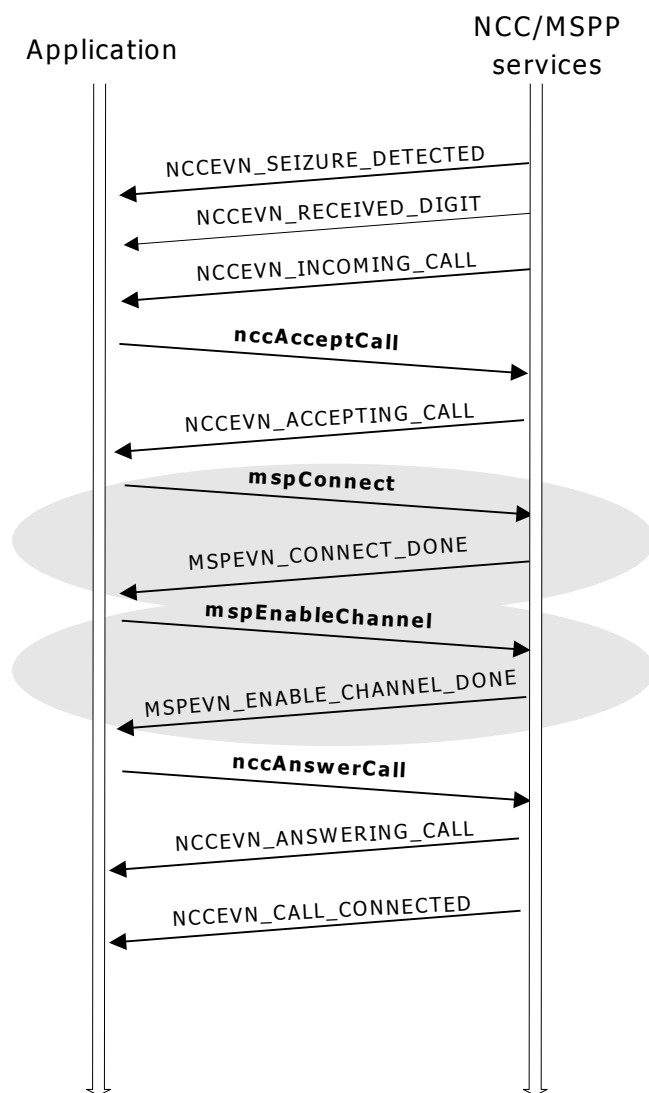
## Receiving inbound calls

In the NCC model for receiving inbound calls, **nccAcceptCall** accepts the incoming call, but does not answer or reject it.

When a Fusion gateway application receives an inbound call from a PSTN, it calls **nccAcceptCall**. After calling **nccAcceptCall**, the application waits for the associated NCCEVN\_ACCEPTING\_CALL event before invoking **mspConnect**. The receiving gateway then waits for the gateway at the opposite end of the network to place the call before the receiving gateway answers it.

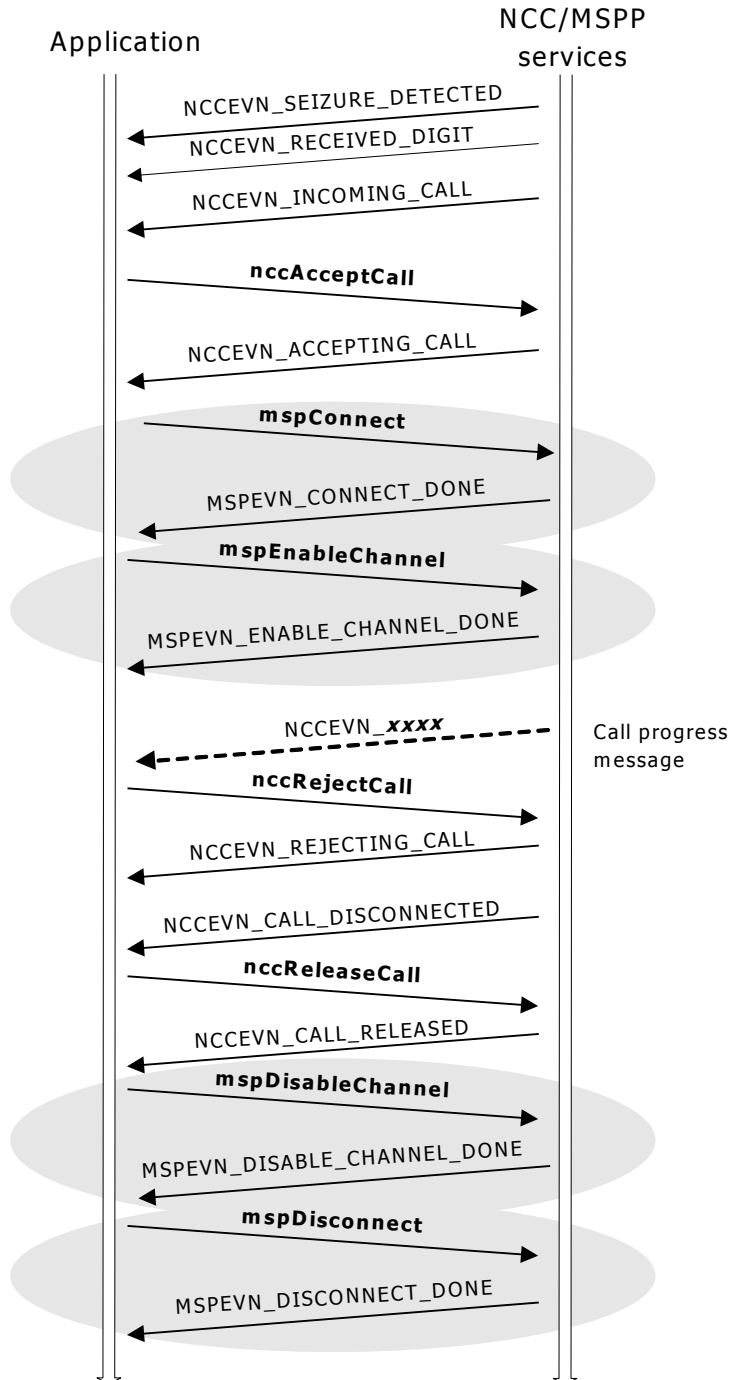
- If the gateway at the opposite end of the network successfully places the call, the receiving gateway answers the call with **nccAnswerCall** and enables the connection with **mspEnableChannel**.
- If the gateway at the opposite end of the network is unsuccessful in placing the call, the receiving gateway rejects the call with **nccRejectCall**.

The following illustration shows function procedures for accepting inbound calls:



When receiving inbound calls, if the gateway application does not use **nccAcceptCall**, it still must wait for the MSPEVN\_CONNECT\_DONE event before answering the call with **nccAnswerCall**. In this case, however, the application cannot monitor the status of the call placed by the gateway at the other end of the network.

The following illustration shows function procedures for rejecting inbound calls:



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

## Placing outbound calls

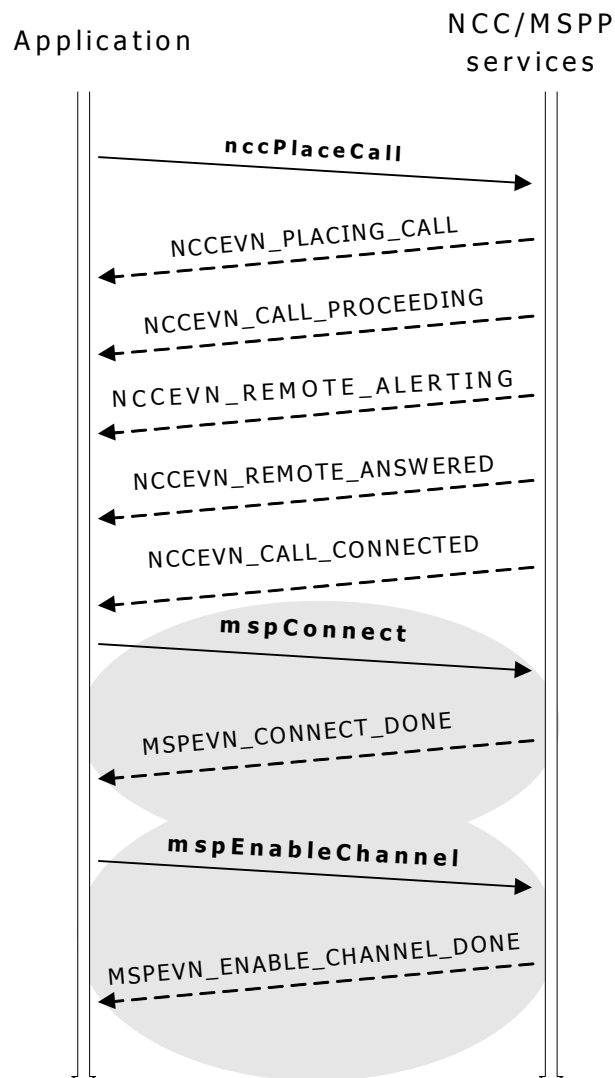
When placing outbound calls, applications use **nccPlaceCall** to place calls to external PSTN networks. Fusion applications wait for the NCCEVN\_CALL\_CONNECTED event (returned by **nccPlaceCall**) before connecting MSPP endpoints and channels with **mvpConnect**.

While waiting for the PSTN to connect the call, the NCC service returns several call progress events that indicate the call's status until the call is answered. The Fusion application placing the outbound call can initiate an MSPP connection during this period in order to play ring back across the IP network that alert the calling party of the status of the call. In this case, the application uses ADI service call progress functions to monitor the call.

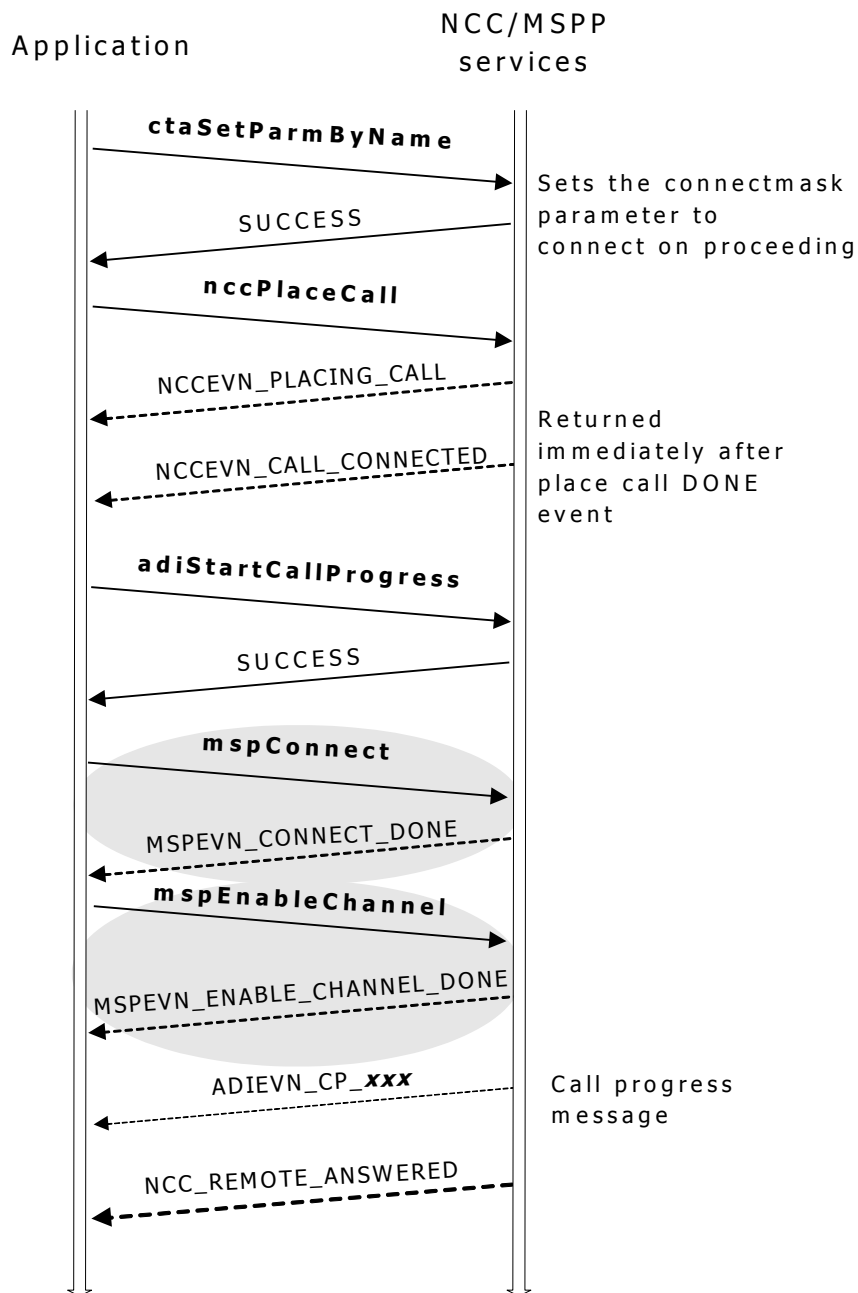
To place an outbound call and to monitor call progress (even after initiating an MSPP connection):

Step	Action
1	Invoke <b>adiStartCallProgress</b> to initiate call progress monitoring by the ADI service.
2	Invoke <b>ctaSetParmByName</b> to set the NCC.X.ADI_PLACECALL.CONNECTMASK parameter to NCC_CON_ON_PROCEEDING. This parameter setting tells the NCC service to return an NCCEVN_CALL_CONNECTED event immediately after it returns an NCCEVN_PLACING_CALL event.
3	Connect an MSPP channel with two MSPP endpoints by invoking <b>mvpConnect</b> . The application can now monitor call progress while playing tones across the IP network that indicate the call's status.

The following illustrations show two sequences for placing outbound calls from a Fusion gateway. In the first sequence, the application does not track the call's progress after the MSPP connection is initiated. The illustrations do not show specific call progress events returned by the ADI service, or other ADI functions called by the application (for example, to play ring back).



The following illustration shows the call setup function sequence as the application monitors call progress from the time the MSPP connection is initiated until the call is answered:



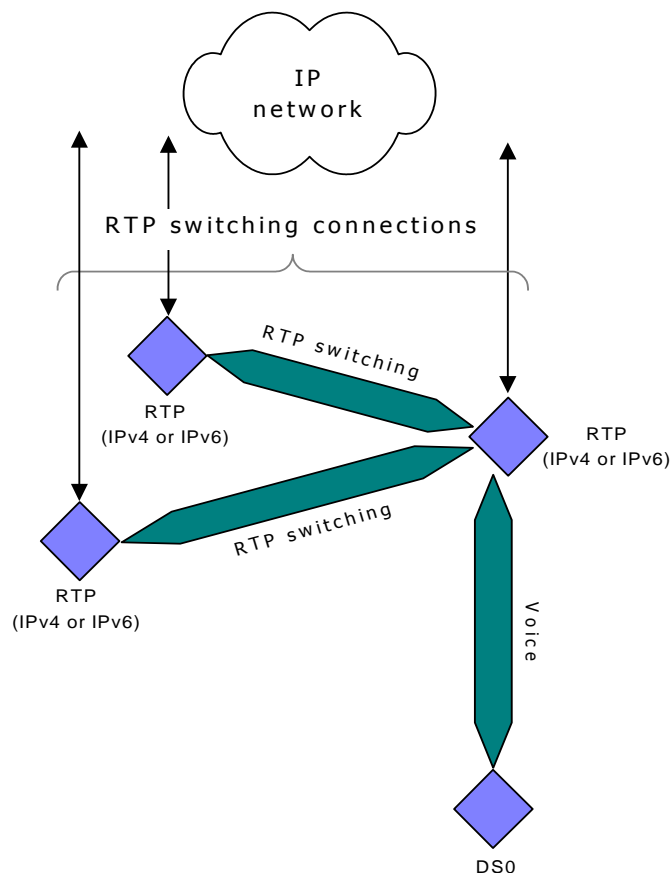
For more information about NCC call control, refer to the *Dialogic® NaturalAccess™ NaturalCallControl™ API Developer's Manual*.

# 7

## Using RTP forking and switching

### RTP forking and RTP switching

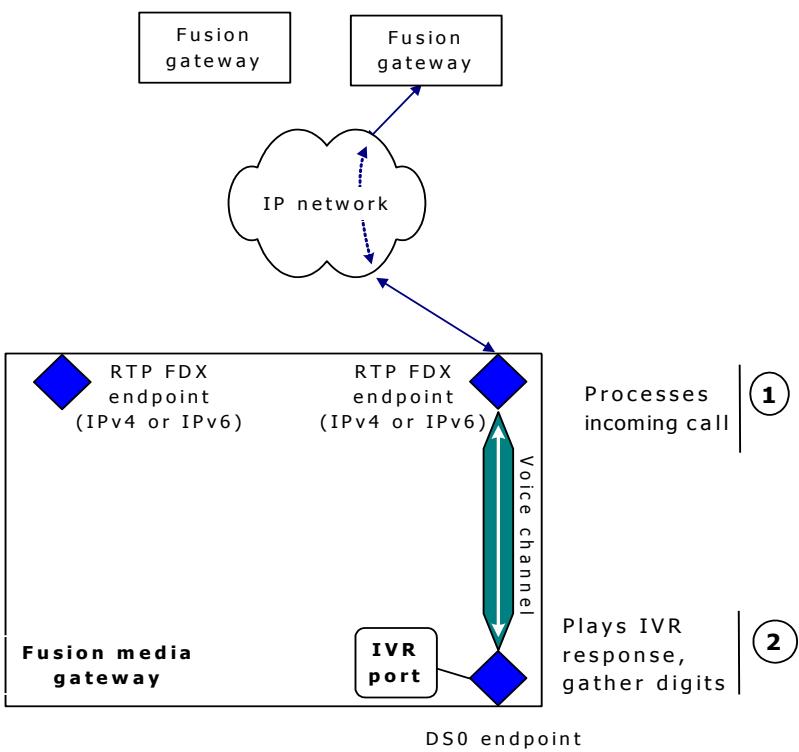
RFC 1889 defines a protocol (RTP) for transporting real time audio and video data over a packet based network. In Fusion Voice over IP gateways, RTP endpoints are generally in point-to-point connections that transfer data between separate points on a network. However, Fusion also provides a mechanism for switching the data from a single RTP IPv4 or RTP IPv6 endpoint to multiple RTP IPv4 or IPv6 endpoints on the same gateway. Directing of a source RTP input stream to another RTP output stream is called RTP switching. Directing a single input RTP stream to multiple output RTP streams is called RTP forking. These capabilities permit applications to add functionality such as support for calling card services and call monitoring. The following illustration shows an example of RTP switching:



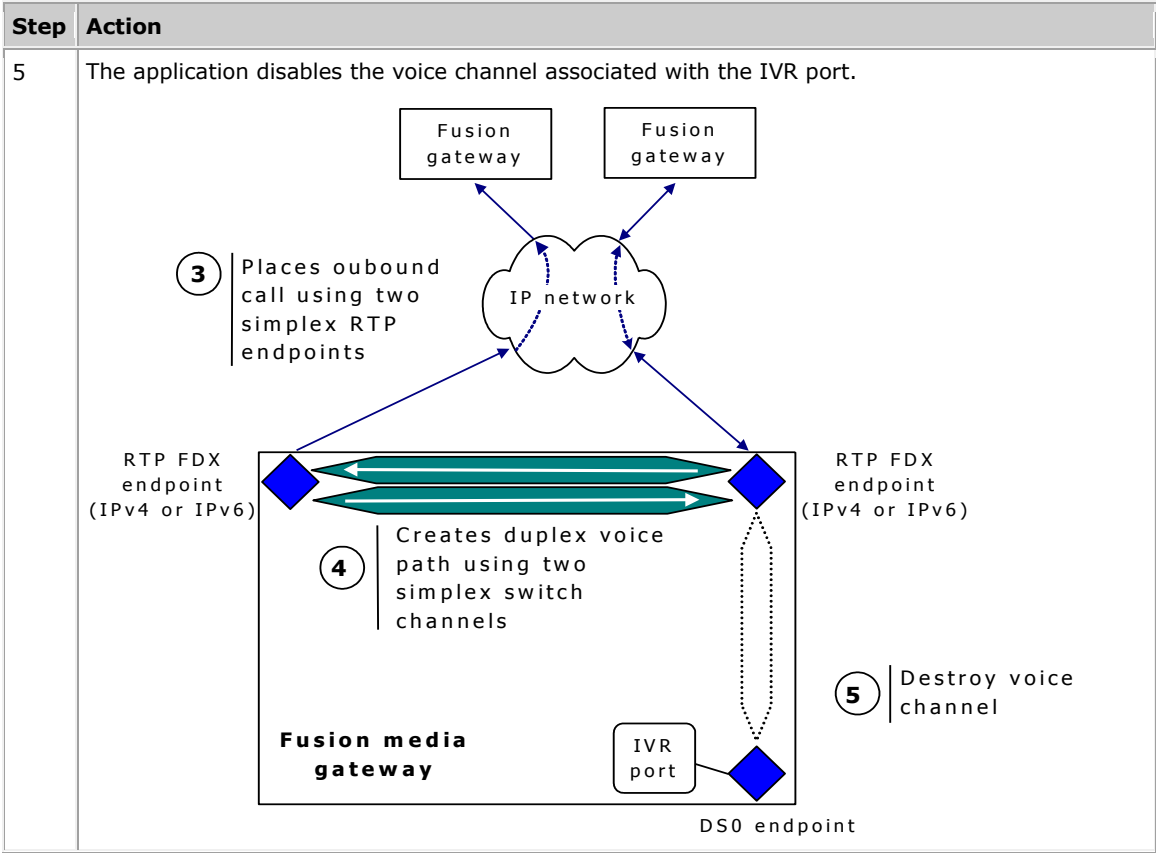
Fusion RTP switching channels consist of a single switch filter used to transfer data between separate RTP endpoints. When applications use an RTP switching channel to connect the two RTP endpoints, the channel transfers the voice packets received on one RTP endpoint to the other RTP endpoint without performing any data conversion tasks.

## Calling card service scenario

Applications that provide calling card services can use RTP switching channels to provide access to these services. A Fusion media server can set up and tear down RTP switching connections to connect calling card users with the outbound calls that they request. In this scenario:

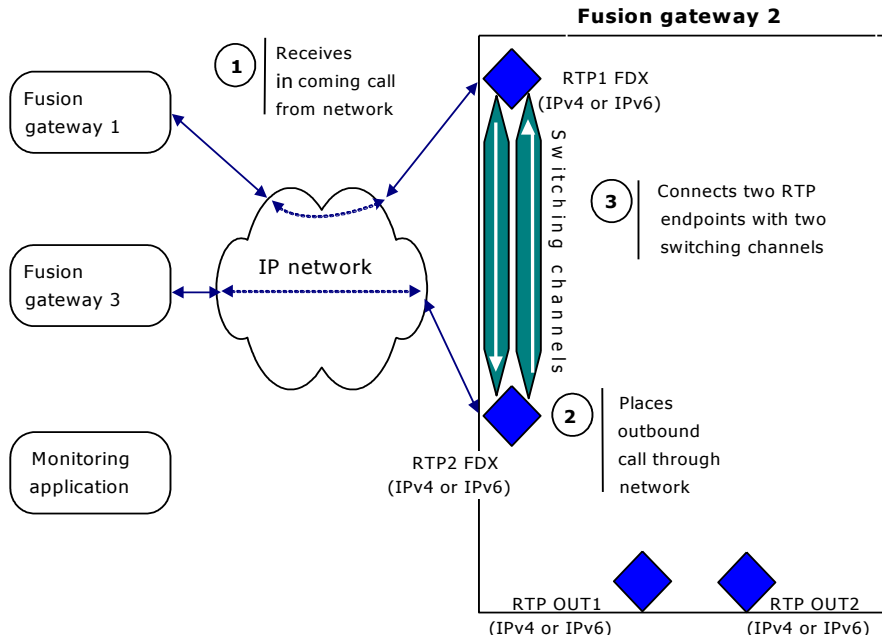
Step	Action
1	The media gateway application receives an incoming call from a calling card user through an RTP endpoint.
2	<p>The media gateway connects the user to an interactive voice response (IVR) port through a voice channel and DS0 endpoint. The application then uses voice message prompts to elicit the number that the user wants to call.</p>  <p>Processes incoming call ①</p> <p>Plays IVR response, gather digits ②</p>
3	The media gateway application places an outbound call to the number that the calling party specified.
4	If the gateway is successful in placing the outbound call, it creates two RTP switching channels to connect the original RTP endpoint with an RTP duplex endpoint associated with the outbound call. These endpoints can be IPv4 endpoints, IPv6 endpoints, or a combination of both.

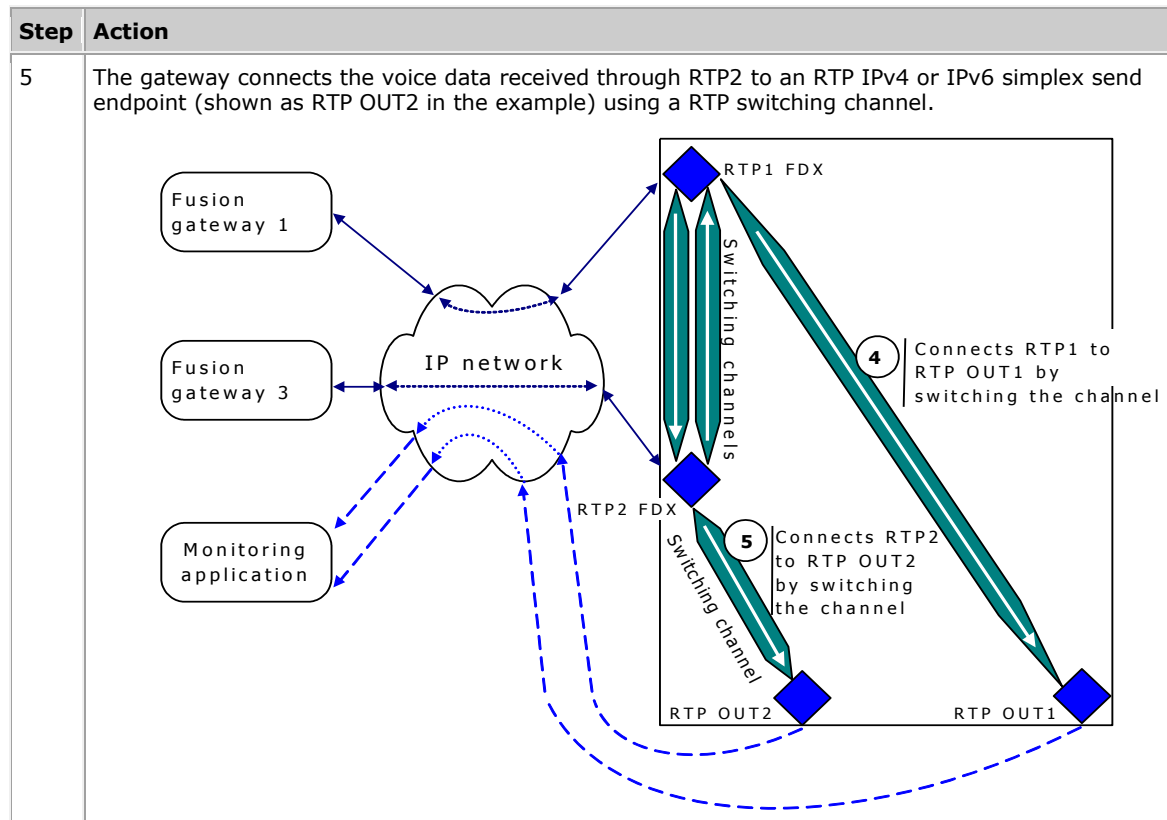




## Monitoring application scenario

Monitoring applications can also use MSPP channel switching to perform legal intercept of RTP streams, to store messages for later playback, or to log connection activity. This feature can be implemented using RTP forking capability. In the example, RTP OUT1 and RTP OUT2 forward voice data to a monitoring gateway on the IP network.

Step	Action
1	A gateway application (Fusion gateway 2) receives an incoming call through an RTP IPv4 or IPv6 duplex endpoint (shown as RTP1 in the example).
2	The gateway places an outgoing call through another RTP IPv4 or IPv6 duplex endpoint (shown as RTP2 in the example).
3	<p>If the gateway is successful in placing the outbound call, it connects the incoming call (RTP1) with the outgoing call (RTP2) using two simplex RTP switching channels.</p>  <p>The diagram illustrates the monitoring application scenario. It shows a central cloud labeled 'IP network'. To the left, there are three boxes: 'Fusion gateway 1', 'Fusion gateway 3', and 'Monitoring application'. To the right, a large box labeled 'Fusion gateway 2' contains several components. At the top, a blue diamond labeled 'RTP1 FDX (IPv4 or IPv6)' is connected to the IP network. Below it, two vertical green bars labeled 'Switching channels' connect the RTP1 FDX to another blue diamond labeled 'RTP2 FDX (IPv4 or IPv6)'. Below the RTP2 FDX, there are two more blue diamonds labeled 'RTP OUT1 (IPv4 or IPv6)' and 'RTP OUT2 (IPv4 or IPv6)'. Three numbered circles with arrows indicate the process: 1. 'Receives incoming call from network' points to the RTP1 FDX. 2. 'Places outbound call through network' points to the RTP2 FDX. 3. 'Connects two RTP endpoints with two switching channels' points to the switching channels. Dashed blue arrows also connect Fusion gateway 1 and Fusion gateway 3 to the IP network.</p>
4	The gateway connects the voice data received through RTP1 to an RTP IPv4 or IPv6 simplex send endpoint (shown as RTP OUT1 in the example) using a RTP switching channel.



## RTP switching limitations

---

When connecting two RTP sessions, consider the following limitations:

- At any given time, there can be only one source of voice data for the switch channel. That is, if the application switches two RTP endpoints that are both receiving voice from DS0 endpoints, it must disconnect one of the RTP endpoints from its associated DS0 endpoint. Otherwise, each RTP endpoint simultaneously receives voice data from two network sources.
- All RTP endpoints switched with switching channels must use the same encoding and decoding algorithm for processing voice data. The switching filter only forwards the packets from one RTP endpoint to another. It does not translate the data from one format to another.
- Each RTP endpoint can be connected to a maximum of 32 switch channels.
- When applications connect both an RTP switching channel and a duplex voice channel to the same RTP and DS0 endpoint, they must connect the duplex voice channel to the RTP endpoint first. RTP endpoints can support more than one RTP switching connection, but can support only one full duplex voice connection whether the connection consists of a single duplex voice channel or two simplex voice channels.

# 8

## Looping back voice data

### Loopback connections

Loopback connections enable applications to monitor the status and quality of voice data transmission within the Fusion gateway. An MSPP loopback receives voice data from a network source, processes the data through a voice channel, and then returns the data to its point of origin (that is, the endpoint from which it was received).

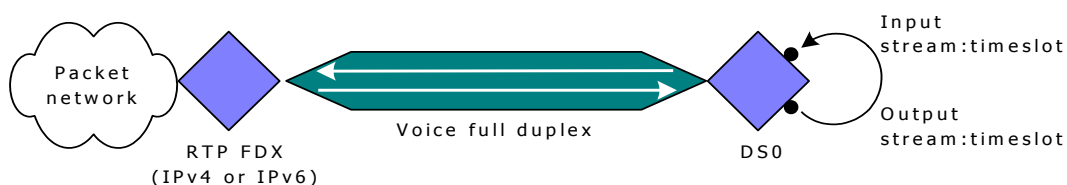
Fusion provides two methods for looping back voice data depending on the type of network (IP or PSTN) from which the data originates:

Loopback	Description
DS0 (PSTN-side)	The gateway receives voice data from an IP network through an RTP endpoint, processes the data through a voice channel, loops the data back at the DS0 endpoint (using the Switching service), and sends the data back to the RTP endpoint that originally received the data.
RTP (IP-side)	The gateway receives voice data from a PSTN through a DS0 endpoint, processes the data through a voice channel, loops the data back at the IP layer (by appropriately configuring the endpoint's local and remote IP address), and sends the data back through the receive side of the IP stack.

In both loopback scenarios, the voice data goes through the full set of voice channel processing. That is, the data is encoded and then decoded, or vice versa.

### DS0 loopback connections

DS0 loopback connections reverse the direction in which voice data flows when the data reaches a DS0 endpoint. Applications use the Switching service to switch the DS0 endpoint's output stream to its input stream and reverse the flow of data back to the connection's RTP IPv4 or IPv6 endpoint. The following illustration shows a DS0 loopback connection:



## RTP loopback connections

RTP loopback connections reverse the direction in which voice data flows when the data reaches an RTP IPv4 or IPv6 endpoint. The data travels through the CG board's IP stack and is looped back at the IP layer so that it can be returned to a DS0 endpoint.

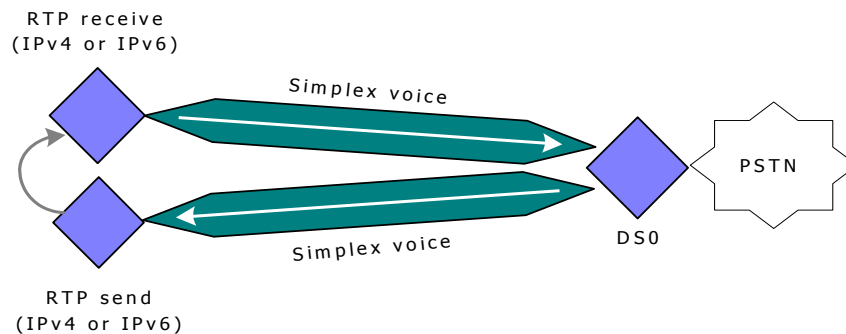
Fusion supports two types of RTP loopback connections:

- Simplex connections
- Duplex connections

### Simplex connections

To loop back voice to the originating DS0, two simplex voice channels must be connected to a single DS0 endpoint and two separate RTP (IPv4 or IPv6) simplex endpoints. In this configuration, the application must set the RTP simplex endpoint's local IP address and remote IP address to the same value. The destination UDP port number of the RTP send endpoint must be the same as the receive UDP port number used by the RTP receive endpoint. The IP address used must be a valid address configured for the CG board and must be associated with an active CG board Ethernet interface.

The application connects the RTP simplex send endpoint with a simplex voice channel and a DS0 endpoint. When the data reaches the RTP send endpoint, it is transferred directly back to the RTP simplex receive endpoint. The RTP simplex receive endpoint then transfers the data through a simplex voice channel back to the DS0 endpoint. The following illustration shows an RTP simplex loopback connection:



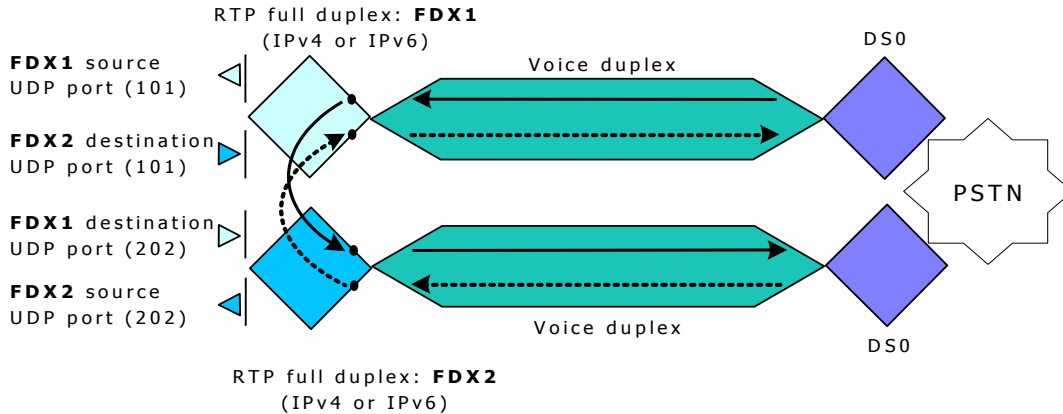
### Duplex connections

The data from an RTP duplex endpoint cannot be looped back upon itself because this causes SSRC collisions and results in dropped packets. However, applications can loop back full duplex data streams at the CG board Ethernet interface by looping together two full duplex voice connections. Each of these connections consists of an RTP IPv4 or IPv6 full duplex endpoint, a full duplex voice channel, and a DS0 endpoint.

When creating RTP duplex loopback connections, the application creates two full duplex voice connections in which the RTP duplex endpoints use different source UDP port numbers. However, each RTP duplex endpoint uses the other endpoint's source UDP port number as its destination UDP port number. In this way, the configuration loops back the two full duplex data streams between the two connections.

For example, an application can create two duplex voice connections in which one of the RTP duplex endpoints (FDX1 in the illustration below) uses 101 as its source UDP port number and the other RTP duplex endpoint (FDX2 in the illustration below) uses 202 as its source UDP port number. To loop back the data from these connections, the application specifies UDP port 202 as the destination port for endpoint FDX1, and specifies UDP port 101 as the destination port of endpoint FDX2. This configuration directs the full duplex voice data output from FDX1 to the input of FDX2, and vice versa.

The following illustration shows an RTP full duplex loopback connection using two full duplex connections:







---

# 9

## Processing MSPP service unsolicited events

---

### Overview of unsolicited events

---

The MSPP service returns a series of unsolicited events that enable applications to monitor the following:

- Route availability of RTP and UDP sessions associated with RTP (IPv4 and IPv6), T38UDP, or TPKT endpoints.
- RTCP reports associated with RTP sessions.
- Inband DTMF carriage status (in accordance with RFC 2833).
- T.38 fax session billing information (described in *Receiving fax billing events* on page 113).

Route availability events are returned automatically if the active data path from an RTP or UDP endpoint is interrupted and then again when the path is reestablished. RTCP and T.38 session unsolicited events are associated with particular types of MSPP channels and must be explicitly enabled by the application.

### RTP and UDP endpoint route availability events

---

Route availability events provide applications with a way to monitor CG board Ethernet interface status transitions that directly affect particular MSPP endpoints. If an application initiates multiple MSPP connections through RTP (IPv4 and IPv6) or UDP endpoints and the Ethernet link remains sound, the application does not receive any route availability events. If a CG board Ethernet interface fails after any RTP or UDP endpoint begins transferring data, and if the boards' ability to route data to a remote RTP or UDP session is affected by that failure, the application receives an MSPEVN\_NOROUTE\_AVAILABLE event. If the Ethernet interface recovers, the application receives an MSPEVN\_ROUTE\_AVAILABLE event.

Unlike the Ethernet link status queries, which are board-level status requests, MSPP route availability events provide status information about specific RTP or UDP sessions. For more information about link status queries, refer to *System-level queries* on page 46. These events can be triggered by a particular Ethernet link failure. However, their main purpose is to provide the application with information about a particular endpoint's ability to successfully transfer data to a remote RTP or UDP session, rather than providing information about the status of a particular Ethernet link.

For example, you can configure CG boards in dual subnet mode so that RTP endpoints are load balanced across both board Ethernet links. In this scenario if one of the Ethernet links fails, the MSPP service only sends route availability events to those RTP endpoints that use the failed link. RTP endpoints associated with the remaining active Ethernet link are unaffected by the failure, and do not receive route availability events.

## Types of unsolicited events

The MSPP service supports the following unsolicited events:

Query	Description
MSPEVN_NOROUTE_AVAILABLE	An RTP or UDP session lost the ability to route outbound data to its destination.
MSPEVN_ROUTE_AVAILABLE	An RTP or UDP session formerly incapable of transmitting data is now back in service.

Route availability events are associated only with data sent over Ethernet connections - not data *received* over these connections. Therefore, these events do not apply to RTP simplex-receive endpoints.

In redundant Ethernet mode, if an RTP endpoint is active and the CG board's primary Ethernet interface fails, the secondary Ethernet interface automatically becomes active, and the RTP endpoint continues to transfer data. When this happens, the application does not receive any route availability unsolicited events, even though the primary Ethernet interface has failed. The application only receives the route availability event if the second CG board Ethernet interface also fails.

Therefore, on a per-session basis, when the CG board is configured in redundant Ethernet mode, a link failure does not necessarily mean that any associated RTP or UDP sessions are interrupted.

When the CG board is configured in dual subnet mode, unsolicited route availability events are returned under different conditions for RTP sessions and UDP sessions.

Endpoint type	Link status event behavior
RTP IPv4 or IPv6	<p>If the CG board Ethernet interface used by a particular RTP IPv4 or IPv6 endpoint fails, the MSPP service returns a route availability unsolicited event to the application informing it that no route is available. This event occurs even if a secondary Ethernet interface is available through another router associated with the other Ethernet interface.</p> <p>This event behavior occurs because the remote side of the RTP session has no way of determining whether the RTP session has been re-routed to another IP address. Even if the RTP IPv4 or IPv6 endpoint re-routes its packets to the second CG board Ethernet interface, it transfers data in only one direction. The remote side has no way of knowing about this redirection, and continues sending data to an unavailable link or IP address.</p>
UDP	<p>If a secondary router is configured for the CG board, route availability events are not returned to the application when the first Ethernet interface goes down. When one of the CG board's Ethernet interfaces goes down, UDP packets are re-routed to the secondary router. The remote UDP application detects and redirects its data packets to the IP address associated with the secondary link.</p> <p>If no secondary router is configured for the CG board when one of the CG board's Ethernet interfaces goes down, the MSPP service returns unsolicited route availability events for all UDP endpoints that are associated with the interface.</p>

Route availability unsolicited events are associated with particular RTP or UDP endpoint handles and do not return any additional buffered data. When the CG board is configured in dual subnet mode, applications receiving MSPEVN\_NOROUTE\_AVAILABLE events can determine which Ethernet interface went down by the endpoints associated with the returned events.

## RTCP report events

---

RTP (real time protocol) is an internet protocol for transmitting real-time data, such as audio and video data, over packet networks. MSPP service RTP IPv4 and IPv6 endpoints support this protocol to provide an entry or exit point for duplex voice-over-IP data in Fusion gateways.

The real time control protocol (RTCP) provides a way to send and receive reports about the information transferred across packet networks in RTP packets. RTCP reports can be sent and received in RTP packets during voice-over-IP transmissions. RTP IPv4 and IPv6 endpoints receive these reports and can (if the application enables this feature) convey the information to the application in the form of unsolicited events.

This topic describes:

- RTCP messages
- Configuring RTP endpoints to return RTCP report information
- Processing RTCP information

## RTCP messages

---

An RTCP message consists of a number of stackable packets, each with its own type code and length indication. The packet format is similar to data packets; in particular, the type indication is at the same location. RTCP packets are sent periodically by the remote gateway, and can serve as a status indicator for session members, even when those members are not actively transmitting media data.

RTCP packets contain the necessary information for quality-of-service monitoring. Applications that have recently sent data generate a sender report. This information allows receivers to estimate the actual data rate.

Session members also periodically issue receiver reports if they have recently received data from remote endpoints. These reports contain the following information:

- The highest packet sequence number received
- The number of packets lost
- A measure of the inter-arrival jitter
- Timestamps needed to compute an estimate of the round-trip delay between the sender and the receiver issuing the report

RTP data packets identify their origin through a randomly generated 32-bit identifier. For conferencing applications, RTCP messages contain an SDES (source description) packet that contains several types of information, usually in text form. One such piece of information is the canonical name, a globally unique identifier of the session participant. Other possible SDES items include the user's name, email address, telephone number, application information, and alert messages.

## Configuring RTP IPv4 and IPv6 endpoints to return RTCP report information

Fusion applications can receive RTCP report information through RTP IPv4 and IPv6 endpoints. By default however, RTP endpoints do not send this information to the remote endpoint. Applications can configure RTP endpoints to return RTCP information by enabling this feature while creating the endpoint, or by commanding the endpoint to return unsolicited events after it is created.

After creating an RTP endpoint, the application can command the RTP endpoints to return RTCP unsolicited events by using **mspSendCommand** with the filter command `MSP_CMD RTPIN_RTCP_EVENTS`. Once commanded to do so, the RTP endpoint periodically sends RTCP report information in the form of unsolicited MSPP events. For more information about creating and commanding RTP endpoints, refer to the *MSPP Service Developer's Reference* manual.

To configure an existing RTP endpoint to return RTCP report information:

Step	Action
1	Create the RTP IPv4 or IPv6 endpoint with <b>mspCreateEndpoint</b> .
2	Disable the endpoint with <b>mspDisableEndpoint</b> . RTP endpoints receive commands only when in a disabled state.
3	Use <b>mspSendCommand</b> to send a <code>MSP_CMD RTPxxx_CONFIG</code> filter command to the endpoint. This filter command requires that you specify an <code>ENDPOINT RTPxxx_CONFIG</code> structure. In this structure, set <code>startRtcp</code> to a non-zero value. The RTP endpoint begins sending periodic unsolicited events when as it receives RTCP reports from the remote endpoint.
4	Use the Natural Access function <b>ctaWaitEvent</b> to wait for the periodic RTCP unsolicited events.
5	Invoke <b>mspReleaseBuffer</b> after the application receives each unsolicited event.  RTCP session information is delivered to the application in a raw form. Fusion provides a set of macros for accessing information stored in bit form within the structure. For more information, refer to the <i>mspunsol.h</i> header file.

For more information about sending commands to MSPP endpoints, refer to the *Dialogic® NaturalAccess™ Media Stream Protocol Processing API Developer's Manual*.

## Processing RTCP information

The unsolicited RTCP events returned by the MSPP service contain buffered RTCP reports received from a remote endpoint. RTP IPv4 and IPv6 endpoints receive the RTCP reports in compound packets that follow a variable format. This format is described in RFC 1889 and subsequent internet drafts. The MSPP service includes this information, in buffered form, in the unsolicited events.

Each buffer within an unsolicited event begins with either a Sender Report or Receiver Report packet (depending on the source), followed by a Source Description packet, and (optionally) a BYE packet. All packet information in the buffer follows a common format. Each packet segment begins with a header in which one field specifies the packet type and another field that specifies its length. The application receiving the buffer can therefore sequentially read information about each packet, starting with the first packet and ending with the last.

For information about RTCP report format and content, refer to RFC 1889. For more information about the specific format of the buffers returned by MSPP unsolicited events, refer to the *mspunsol.h* header file.

## RFC 2833 related events

When the application enables inband DTMF detection, MSPP voice channels generate the following events when they detect DTMF tones:

Event	Description
MSPEVN_DPF_DTMF_REPORT	Returned when a voice encoder filter detects a DTMF digit on a data stream received from a PSTN.
MSPEVN_DPF_DTMF_PLAY_END	Returned when a voice decoder filter finishes generating a DTMF digit (as directed by a MSP_CMD_DECODE_PLAY_DTMF_DIGIT filter command).
MSPEVN_RFC2833_REPORT	Returned when the RTP endpoint receives an RFC 2833 packet.

MSPEVN\_RFC2833\_REPORT events are returned both when a voice encoder filter detects a DTMF tone, and when an RTP endpoint detects an inband DTMF packet. In both cases the events return DTMF tone information in the following structure:

```
typedef struct {
    DWORD   FilterID;
    U8      EvtID;
    U8      EvtVol;
    WORD    EvtDuration;
}DISASM_DTMF_EVENT_STRUCT;
```

Fields in the returned structure provide the following information:

Field	Description
FilterID	Filter ID associated with the voice encoder filter in the MSPP channel that detected the DTMF tone.
EventID	Detailed information about the type of DTMF detected. Possible values include: 0-9:DTMF digits 0 - 9, respectively 10: DTMF digit * 11: DTMF digit # 12-15: DTMF digits A - D, respectively 16: DTMF digit End notification
EvtVol	For DTMF digits and other events representable as tones, this describes the power level of the tone, expressed in dB. Power levels range from 0 to -63 dB. The range of valid DTMF is from 0 to -36 dBm0 (must accept); lower than -55 dBm0 must be rejected (TR-TSY-000181, ITU-T Q.24A). Therefore, larger values denote lower volume. The EvtVol value is defined only for DTMF digits. For other events, it is set to zero by the sender and is ignored by the receiver. For more information refer to RFC 2833.
EvtDuration	Duration of the digit, in timestamp units. The digit begins at the instant identified by the RTP timestamp and lasts as long as indicated by this value. For sampling rates of 8000 Hz, this field is can express event durations of up to approximately 8 seconds. For more information refer to RFC 2833.

When the Fusion application at the receiving gateway receives an unsolicited event indicating that the MSPP channel has detected a DTMF tone, the application can generate a DTMF tone for the receiving party using the ADI service. Fusion applications must invoke the MSPP service function **mSPReleaseBuffer** after receiving DTMF carriage unsolicited events.



---

# 10 Eliciting board information through MSPP queries

---

## Board-level queries and unsolicited events

---

The MSPP service supports a set of queries and unsolicited events associated with board resources rather than with specific MSPP endpoint filters. These allow applications to use **mspSendQuery** to monitor the status of board-level CPU resources and Ethernet connections while the application is running.

The MSPP service supports the following board-level queries:

Query	Description
MSP_QRY_ETHERNET_LINK_STATUS	Returns information about the board's Ethernet link connections.
MSP_QRY_CPU_USAGE	Returns information about the board's current and average CPU utilization.

When an application uses either of these queries, it can specify any valid MSPP handle (endpoint or channel) associated with the board. However, when using **mspBuildQuery** to build the query, the application must specify the constant **MSP\_SYSTEM** as the filter ID for the query. The following example shows how to build an Ethernet link status board query using the **MSP\_SYSTEM** constant as the filter ID:

```
ret = mspSendQuery(pVoiceChannelDbase[0]->hDs0MspHd,
mspBuildQuery(MSP_SYSTEM, MSP_QRY_ETHERNET_LINK_STATUS) );
if ( ret == FAILURE)
{
    printf("Msp Send Query failed \n");
}
```

In addition to using MSPP service queries, applications can use route availability events to monitor state transitions of CG board Ethernet interfaces.

## Ethernet link status queries

Ethernet link status queries return information about the Ethernet connections that the CG board maintains. MSP\_QRY\_ETHERNET\_LINK\_STATUS queries return the following structure:

```
typedef struct {
    DWORD      numETHs;
    ETH_STATUS  eth[MAX.CG_ETH];
} msp_ETH_STATUS;
```

The msp\_ETH\_STATUS structure specifies the number of active board Ethernet connections (in the numETHs field), and provides the following substructure for each active Ethernet connection:

```
typedef struct {
    DWORD      ethStatus;
    DWORD      ethMode;
    DWORD      ethSpeed;
} ETH_STATUS;
```

Each ETH\_STATUS substructure provides the following information:

Field	Description
ethStatus	Indicates whether the Ethernet connection is up (ETH_LINK_UP) or down (ETH_LINK_DOWN).
ethMode	Indicates whether board Ethernet connection is a half duplex (ETH_HALFDUPLEX) or full duplex (ETH_FULLDUPLEX).
ethSpeed	Specifies the board's Ethernet throughput capacity as 10 Mb (ETH_10Mb) or 100 Mb (ETH_100Mb).

## CPU utilization queries

CPU utilization queries return information about available CG board CPU resources. MSP\_QRY\_CPU\_USAGE queries return the following structure:

```
typedef struct {
    DWORD      numCPUs;
    CPU_UTIL    cpu[MAX.CG_CPU];
} msp_CPU_UTIL;
```

This structure indicates the number of CPUs resident on the board (in the numCPUs field), and provides the following substructure for each active board CPU:

```
typedef struct {
    DWORD      currentUtil;
    DWORD      averageUtil;
} CPU_UTIL;
```

Each CPU\_UTIL substructure provides the following information:

Field	Description
currentUtil	Current CG board CPU utilization (sampled approximately once per second) expressed as percentage of the maximum possible utilization.
averageUtil	Average utilization over the last 16 seconds.



## Route availability events

The MSPP service supports the following session-level unsolicited events (also called route availability events):

Query	Description
MSPEVN_NOROUTE_AVAILABLE	An RTP or T38UDP endpoint lost the ability to route outbound data to its destination.
MSPEVN_ROUTE_AVAILABLE	An RTP or T38UDP endpoint formerly incapable of transmitting data is now back in service.

Route availability events provide applications with a way to monitor CG board Ethernet interface state transitions. By default, these events are disabled when the application creates the endpoint. If the application initiates multiple MSPP connections (through RTP or T38UDP endpoints) and the Ethernet link remains sound, the application never receives a route availability event. However, if the Ethernet interface fails after any RTP IPv4 or IPv6 duplex or simplex-send or T38UDP endpoint begins transferring data, the application receives an MSPEVN\_NOROUTE\_AVAILABLE event. If the Ethernet interface recovers, the application receives an MSPEVN\_ROUTE\_AVAILABLE event.

Route availability events are associated only with data sent over Ethernet connections, not data received over these connections. Therefore, these events do not apply to applications that use only RTP simplex-receive endpoints. Because no data is transmitted, no event is returned even if the Ethernet connection goes down.



---

# 11 Implementing RFC 2833 support

---

## Transferring DTMF digits according to RFC 2833

---

RFC 2833 (*RTP Payloads for DTMF Digits, Telephony Tones, and Telephony Signals*) specifies an RTP payload format for carrying dual-tone multi frequency (DTMF) digits, and other line and trunk signals. Fusion channels that support RFC 2833 provide the following features:

- Detect the presence of DTMF digits in voice data received from PSTNs and pass this information via RFC 2833 compliant RTP packets for the duration of the DTMF digit.
- Send RFC 2833 packets for a specified duration. RFC 2833 specifies a packet format for DTMF digits, line events, and trunk events.
- Detect incoming RFC 2833 RTP packets, and send notifications to the host. RFC 2833 RTP packets that contain DTMF information are passed down the filter chain to be played out to the PSTN interface.

Fusion supports transferring DTMF digits according to RFC 2833 in both RTP IPv4 and RTP IPv6 endpoints.

## Transferring DTMF digits with RFC 2833

---

RFC 2833 specifies a payload format for carrying DTMF signaling information within RTP packets. Complying with the RFC 2833 standard provides the following advantages:

- Reduces the risk of low bit-rate vocoders such as G.723.1 rendering DTMF tones unintelligible to receiving gateways.
- Makes it unnecessary for VoIP gateways to perform DTMF detection on incoming voice streams. Instead, applications can detect DTMF tones by waiting for specially formatted inband DTMF packets.

Fusion implements the RFC 2833 specification by enabling MSPP voice channels to detect DTMF digits in voice data received from PSTNs (that is, through DS0 endpoints). Voice channels then package the DTMF tone information into specially formatted RTP packets before transferring them to a packet network. These inband DTMF packets do not contain any voice data. The payload contained in each packet payload only contains information about the type of tone detected, the tone's gain, and the tone's duration.

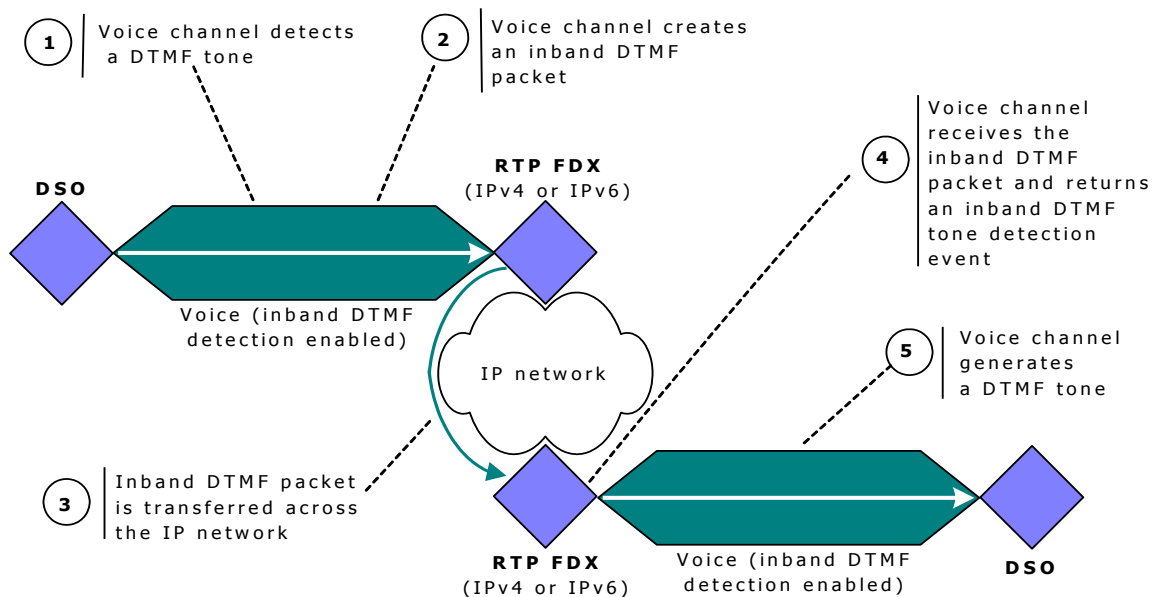
In a gateway that receives the RFC 2833 compliant inband DTMF packet, the MSPP voice channels detect the packets. When the voice channel on the receiving gateway application receives the RFC 2833 compliant payload, it sends the application an event indicating the digit received and generates an appropriate tone based on the information within the packet payload. For a description of MSPP unsolicited events associated with connections configured to support RFC 2833, refer to *RFC 2833 related events* on page 69.

The receiving gateway does not use other services (such as the ADI service) to detect DTMF digits in the incoming voice data stream received from the packet network, or to generate DTMF tones for data directed to the PSTN.

The Fusion inband DTMF mechanism method works in the following way:

Step	Action
1	The MSPP voice channel on the Fusion gateway receives a DTMF tone through the PSTN.
2	The channel generates RFC 2833 compliant inband DTMF packets for the detected tone until the tone ceases.
3	The gateway continuously generates inband DTMF packets and transfers them to the IP network.
4	When another Fusion gateway receives an RFC 2833 compliant inband DTMF packet (through an RTP IPv4 or IPv6 endpoint), the MSPP service reads the payload content. The MSPP service generates an unsolicited event that specifies the identity, gain, and duration of the received DTMF tone.
5	The MSPP voice channel also generates the appropriate DTMF tone on the outgoing voice data stream.

The following illustration shows how to implement inband DTMF carriage through RFC 2833 compliant packets:



## Transferring non-DTMF RFC 2833 events

Applications can send any DTMF, line, and trunk events specified in RFC 2833 by sending the `MSP_CMD_ENCODE_PLAY_RFC2833` command to the voice encoder filter.

This command specifies RFC 2833 line or trunk event and its duration. The voice encoder filter sends this information in RFC 2833 compliant RTP packets. When using this command, applications send a structure containing the following fields:

Field	Type	Description
event	WORD	RFC 2833 event (0 - 255) to include within the packet payload. Refer to the RFC 2833 specification for information about available options.
duration	WORD	Duration of the RFC 2833 event, in milliseconds.

## Setting up RFC 2833 capabilities

To implement RFC 2833 compliant inband DTMF carriage, perform the following tasks:

Step	Action
1	Specify an appropriate Resource keyword string in the CG board keyword file that appropriately configures the board resources needed to support RFC 2833 capabilities.
2	Start a protocol on the context that is associated with the voice channel, and specify appropriate mediamask parameters to support DTMF detection in the voice channel instead of the DS0 endpoint.
3	Specify settings in the MSPP channel address and parameter structures at voice channel create time that enable and configure RFC 2833 capabilities.

Before you implement RFC 2833 compliant inband DTMF carriage on a particular CG board, specify Resource settings in the associated board keyword file that identify the necessary board resources. The following example shows an OAM board keyword string that specifies DPFs to support RFC 2833 compliant inband DTMF carriage (shown in **bold**). These DPFs are combined using the OR operator with DPFs that do not support inband RFC 2833 capabilities so that the board can support both modes of operation.

```
Resource[0].Definitions = (f_echo_v3.ln20_apt25 & \
  ((dtmf.det_sil_clrdown_ced & \
    ((f_g711.cod_rfc2833 & f_g711.dec_rfc2833) | \
    (f_g729a.cod_rfc2833 & f_g729a.dec_rfc2833) | \
    (f_g723.cod_rfc2833 & f_g723.dec_rfc2833) | \
    f_faxt38.relay))) | \
  (dtmf.det_all & \
    ((f_g711.cod & f_g711.dec) | \
    (f_g729a.cod & f_g729a.dec) | \
    (f_g723.cod & f_g723.dec) | \
    f_faxt38.relay)))) |
```

As the example shows, the DPFs that support RFC 2833 capabilities end with the string **`_rfc2833`**. When you use `oamsys` to configure the board, the CG board resource manager calculates resources based on two different scenarios: RFC 2833 capabilities enabled (**bold**), or RFC 2833 capabilities disabled. For more information about CG board resource management, refer to *Configuring CG board resources* on page 31 or to the CG board manual.

Before you can create an MSPP channel create a Natural Access context with **ctaCreateContext**. When enabling RFC 2833 compliant inband DTMF carriage, you must also use **nccStartProtocol** to start a protocol on the context. When invoking **nccStartProtocol**, specify NCC parameters that reserve DSP resources required for call control operations performed on that context. This determines the functionality available for calls associated with the context.

To implement inband DTMF carriage, specify bits for the NCC.X.ADI\_START.mediamask parameter that enable clear down detection, silence detection, and echo cancellation. You can specify appropriate mediamask parameters by using any (or a combination of) the following mediamask settings:

Bit setting	Description
NCC_CC_RESVSILENCE1	Reserves silence detection.
NCC_CC_RESVCLRDWN	Reserves clear down detection.
NCC_CC_AUTOECHO	Starts echo canceller.

For example, the following sample code shows how to define a flag that specifies appropriate mediamask settings to support RFC 2833 compliant inband DTMF carriage:

```
if (pCmdLineParms->bEnableInBandDtmf)
{
    startparms.mediamask =
        NCC_CC_RESVSILENCE | NCC_CC_RESVCLRDWN | NCC_CC_AUTOECHO;
    /* Mediamask for inband DTMF carriage (RFC 2833) */
}
else
{
    startparms.mediamask =
        NCC_CC_RESVDTMF | NCC_CC_RESVSILENCE | NCC_CC_RESVCLRDWN |
        NCC_CC_AUTODTMF | NCC_CC_AUTOECHO;
    /* Mediamask for conventional DTMF carriage */
}
```

## Configuring channels for RFC 2833 support

Applications create MSPP service channels by invoking **mspCreateChannel**. By default, MSPP channels do not implement RFC 2833 compliant inband DTMF carriage. To create channels that support RFC 2833 capabilities, the application must enable these capabilities with the `FilterAttribs` parameter in the channel address structure, and configure how the capabilities are implemented with parameters in the channel parameters structure.

When applications create MSPP channels, they specify the following channel address parameters in the following structure:

```
typedef struct tag_MSP_CHANNEL_ADDR
{
    DWORD          size;
    DWORD          nBoard;
    MSP_CHANNEL_TYPE channelType;
    WORD           FilterAttribs;
} MSP_CHANNEL_ADDR;
```

By setting the `FilterAttribs` parameter in the channel address structure to `MSP_FCN_ATTRIB_RFC2833`, the application enables RFC 2833 capabilities for the channel. The following code sample shows how to specify the `FilterAttribs` parameter at channel create time to have the channel support RFC 2833 capabilities:

```
MSP_CHANNEL_ADDR    MspVoiceChanAddr

if (EnableInBandDtmf)
    MspVoiceChanAddr.FilterAttribs = MSP_FCN_ATTRIB_RFC2833;
else
    MspVoiceChanAddr.FilterAttribs = 0;

mspCreateChannel(ctahandle, &MspVoiceChanAddr, &MspVoiceChannelParms, MspHandle)
```

If the application does not enable RFC 2833 capabilities with the `FilterAttribs` parameter when it creates an MSPP channel, the functionality cannot be enabled on the channel at a later time.

The `FilterAttribs` parameter enables RFC 2833 capability support on the channel according to default settings. The default settings specify the following functionality:

Data path	Default MSPP channel processing for inband DTMF
Voice data received from PSTN	<ul style="list-style-type: none"> <li>• Detects DTMF tones in voice data streams received from DS0 endpoints.</li> <li>• Transfers DTMF tone information only through RFC 2833 compliant DTMF packet streams (that is, it does not also send the data as encoded voice).</li> <li>• Does not generate <code>MSPEVN_DPF_DTMF_REPORT</code> unsolicited events when it detects DTMF tones.</li> </ul>
Voice data received from IP network	<ul style="list-style-type: none"> <li>• Receives RFC 2833 compliant packets.</li> <li>• Generates events indicating the tone detected.</li> </ul>

## Customizing RFC 2833 compliant channel behavior

When creating an MSPP channel, applications define settings in a `MSP_VOICE_CHANNEL_PARMS` structure to customize the functionality associated with the created channel. This structure includes two `DtmfMode` parameters that specify how inband DTMF carriage works on the channel.

This topic describes:

- Setting encoder RFC 2833 parameters
- Setting decoder RFC 2833 parameters
- Playing tones manually through a voice decoder filter

Each `DtmfMode` parameter resides in the separate encoder or decoder substructure within the `MSP_VOICE_CHANNEL_PARMS` structure as shown in the following example:

Encoder substructure	Decoder substructure
<pre>typedef struct tag_msp_FILTER_ENCODER_PARMS {     DWORD size;     WORD Mode;     WORD Gain;     WORD VadControl;     WORD NotchControl;     WORD IPFormat;     WORD Rate;     WORD PayloadID;     WORD DtmfMode; } msp_FILTER_ENCODER_PARMS;</pre>	<pre>typedef struct tag_msp_FILTER_DECODER_PARMS {     DWORD size;     WORD Mode;     WORD Gain;     WORD IPFormat;     WORD PayloadID;     WORD DtmfMode; } msp_FILTER_DECODER_PARMS;</pre>

### Setting encoder RFC 2833 parameters

Each `DtmfMode` parameter within the `msp_FILTER_ENCODER_PARMS` substructure is a 16-bit `WORD` made up of two 8-bit bytes. These bytes specify whether the channel detects DTMF tones, how it passes on this information (as voice data, in DTMF packets, or both), and/or whether the application generates unsolicited events when it detects DTMF tones.

The following table describes the two bytes (Control and PayloadID) that define MSPP channel encoder inband DTMF carriage functionality:

Byte	Available settings
Control	<ul style="list-style-type: none"> <li>• <code>DTMF_INBAND_ENABLED</code>: Encoder detects DTMF tones and transfers DTMF information through RFC 2833 compliant inband DTMF packets.</li> <li>• <code>DTMF_EVENTS_ENABLED</code>: Encoder generates unsolicited events when it detects DTMF tones. This setting is disabled by default.</li> <li>• <code>VOICE_ENABLED</code>: Encoder sends DTMF tones as voice data. In this case, <code>DTMF_INBAND_ENABLED</code> must also be set.</li> <li>• <code>DTMF_SHIFT_ENABLED</code>: If set, the encoder shifts the timestamp of the associated DTMF RTP packets back to more accurately align it with the start of DTMF detection. This shift allows for more synchronized decoder playout of RFC 2833 DTMF digits with respect to actual DTMF time.</li> </ul>
PayloadID	Encoder sets the payload ID for RFC 2833 compliant inband DTMF packets. Range is 96-127. Default value is 96.



Control parameter settings are not mutually exclusive. That is, applications can combine multiple settings to combine different kinds of functionality.

For information about processing the unsolicited events associated with RFC 2833 capabilities, refer to RFC 2833 related events.

### Setting decoder RFC 2833 parameters

Each DtmfMode parameter within the msp\_FILTER\_DECODER\_PARMS substructure is a 16-bit WORD made up of two 8-bit bytes. These bytes specify whether the channel plays DTMF tones when it receives RFC compliant inband DTMF carriage reports.

The following table describes the two bytes (control and playoutvalue) that define channel decoder inband DTMF carriage functionality:

Byte	Available settings
control	<ul style="list-style-type: none"> <li>DTMF_DISABLED: Decoder does not play a DTMF tone when it receives an RFC 2833 compliant inband DTMF carriage packet.</li> <li>DTMF_INBAND_ENABLED: Decoder automatically plays a DTMF tone when it detects an RFC 2833 compliant inband DTMF carriage packet. This setting is enabled by default.</li> <li>DTMF_PLAY_ENABLED: Decoder plays DTMF tones when the application sends an MSP_CMD_DECODE_PLAY_DTMF_DIGIT command.</li> </ul>
playoutvalue	Specifies an integer value that indicates the number of decoder frames that the MSPP service generates before stopping when no end-of-tone packet is received. The default value is 3.

Control parameter settings are not mutually exclusive. For example, applications that perform out-of-band DTMF detection can combine DTMF\_DISABLED with DTMF\_PLAY\_ENABLED. These settings stop the channel from detecting RFC 2833 compliant packets while still allowing the application to generate DTMF tones through the MSPP channel.

### Playing tones manually through a voice decoder filter

Applications that generate out-of-band DTMF tones can play DTMF tones through the voice channel by sending MSP\_CMD\_DECODE\_PLAY\_DTMF\_DIGIT commands to the decoder filter associated with an MSPP channel. When the MSPP channel decoder filter receives this command, it plays the specified tone. When sending this command, the application includes a structure with a Digit and Duration fields. These fields provide information about the DTMF digit to play. When the channel finishes playing the tone, it returns an MSPEVN\_DPF\_DTMF\_PLAY\_END event. This event informs the application when a complete tone has been played.

## Configuring RFC 2833 settings

RFC 2833 capabilities are implemented through both MSPP voice channels and RTP IPv4 and IPv6 endpoints. The FilterAttribs parameter that the application specifies when it creates an MSPP voice channel indicates how the channel reacts when it receives RFC 2833 compliant packets through a DS0 endpoint. In addition, RTP IPv4 and IPv6 endpoint default settings support RFC 2833 capabilities. The following table provides details about the default RFC 2833 compliance settings for voice channels and RTP IPv4 and IPv6 endpoints:

MSPP object	Description
Voice channel	Encoder - transfers DTMF tones as RFC 2833 compliant packets but not as voice data, and does not generate unsolicited MSPEVN_DPF_DTMF_REPORT events at the beginning and end of each tone.  Decoder - generates appropriate DTMF tones when it receives RFC 2833 compliant packets.
RTP IPv4 and IPv6 endpoints	Receives RFC 2833 compliant packets and generates MSPEVN_RFC2833_REPORT events at the beginning and end of the packet stream associated with each RFC 2833 event.

To specify how RFC 2833 capabilities are supported on an MSPP channel, the application sends MSPP filter commands to the voice channel. Applications can send the following commands to MSPP channels on which RFC 2833 capabilities are enabled:

Command	Filter	Description
MSP_CMD_ENCODE_DTMF_MODE	Voice encoder	Specifies a DtmfMode parameter for the encoder filter, allowing the application to re-configure how inband DTMF carriage works.
MSP_CMD_DECODE_DTMF_MODE	Voice decoder	Specifies a DtmfMode parameter for the decoder filter, allowing the application to re-configure how inband DTMF carriage works.

For more information about sending commands to MSPP endpoint and MSPP channel filters, refer to the *Dialogic® NaturalAccess™ Media Stream Protocol Processing API Developer's Manual*.

By default, RTP IPv4 and IPv6 endpoints support RFC 2833 capabilities. That is, when an application creates an MSPP channel that supports inband DTMF carriage, the connected RTP endpoint automatically generates appropriate events when it receives RFC 2833 compliant packets.

## Customizing RTP endpoint behavior

An application can customize how RTP endpoint RFC 2833 behavior by:

- Configuring RTP endpoint parameters when creating the RTP IPv4 or IPv6 endpoints
- Sending an MSPP endpoint command to an RTP IPv4 or IPv6 endpoint

### Configuring RTP endpoint parameters

Applications configure RTP endpoint RFC 2833 behavior when they create RTP IPv4 and IPv6 endpoints. The RTP endpoint parameter structure (within the RTPRTCP\_ENDPOINT\_PARMS structure) includes a rfc2833\_event\_control parameter for configuring how the RTP IPv4 or IPv6 endpoint notifies the application when it receives an RFC 2833 compliant packet. Each rfc2833\_event\_control parameter is a 32-bit DWORD made up of two 16-bit WORDS. The following table describes the two WORDS that define RTP IPv4 and IPv6 endpoint inband RFC 2833 functionality:

WORD	Available settings
dtmf_event_control	<ul style="list-style-type: none"> <li>• SEND_ALL_EVENTS: Endpoint generates an unsolicited event for every inband RFC 2833 compliant packet it receives.</li> <li>• SEND_FIRST_EVENT: Endpoint generates an unsolicited event when it receives the first RFC 2833 compliant packet associated with a DTMF tone, or another trunk, or line signal.</li> <li>• SEND_LAST_EVENT: Endpoint generates an unsolicited event when it receives the last RFC 2833 compliant packet associated with a DTMF tone, or another trunk, or line signal.</li> <li>• SEND_NO_EVENTS: Endpoint generates no unsolicited events.</li> </ul> <p><b>Note:</b> SEND_FIRST_EVENT and SEND_LAST_EVENT are enabled by default.</p>
decimation (used in conjunction with SEND_ALL_EVENTS)	Specifies how often the RTP endpoint sends DTMF events for a particular tone by specifying how many packets to ignore for a given tone before generating another event. For example, specifying a value of 5 causes the RTP endpoint to generate an inband DTMF tone event for every fifth RFC 2833 compliant packet received for a given DTMF tone.

Use the OR operator to combine dtmf\_event\_control values.

### Sending RTP IPv4 and IPv6 endpoint filter commands

Applications can also send MSP\_CMD RTPIN\_DTMF\_EVENTS commands to RTP receive endpoint and RTP endpoints (IPv4 and IPv6) that specify how the endpoint generates events when it receives RFC 2833 compliant DTMF packets. The MSP\_CMD RTPIN\_DTMF\_EVENTS command requires that the application specify the same dtmf\_event\_control parameter that is included in the RTP endpoint parameter structure. The created RTP endpoint then generates inband DTMF unsolicited events according to what the application specifies with the rfc2833\_event\_control parameter.

For more information about sending commands to MSPP endpoint and MSPP channel filters, refer to the *Dialogic® NaturalAccess™ Media Stream Protocol Processing API Developer's Manual*.



---

# 12 Using native play and record

---

## Performing NMS native play and record

---

NMS native play and record enables applications to maintain the quality of audio data played and recorded over network interfaces while minimizing the encoding and decoding resources needed to process the audio data.

Applications can use the MSPP API to perform the following tasks with NMS native play and record:

- Record voice data from RTP data streams transferred through MSPP service endpoints.
- Play media recorded directly from RTP streams to PSTN (DS0) ports.
- Play and record media streams that contain silence, SID frames, RFC 2833 markers, and lost frame markers.
- Perform silence and DTMF detection while recording decoded RTP streams.

CG boards support NMS native play and record functionality.

## NMS native play and record advantages

---

When an application plays or records audio data over a TDM or IP network, typically the application must encode or decode the data. In TDM networks, audio data is normally coded in either mu-law or A-law format. In IP networks, audio data is often encoded in a compressed format such as G.711 or G.723.1. Encoding or decoding the audio stream can consume system resources and incrementally degrade the quality of the data.

When an application records audio data using native record, the audio is stored in the NMS EDTX (extended discontinuous transmission) format without encoding the data. The application can then either play the audio data directly to a DS0 or network interface or transfer the data to the interface through an encoder or decoder.

## Implementing NMS native play and record

---

NMS native play and record uses an NMS proprietary format called EDTX (extended discontinuous transmission) to store and play back codec frames to and from RTP endpoints. EDTX formatting incorporates an optional silence compression scheme that uses silence frames in the recorded stream to indicate periods of silence.

Native play and record supports the following encoding types:

- G.711A, G.711U
- G.723.1
- G.726
- G.729A/B

For more information about supported vocoder types, refer to the Fusion vocoder *readme.txt* files.

Applications can implement native play and record in the following ways:

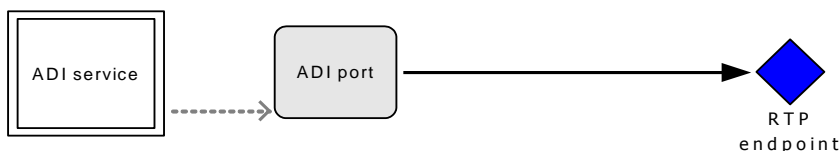
Implementation	Description
Native play	Application plays a stream of audio data from an ADI port to an RTP endpoint.
Native record with inband silence and DTMF detection	Application receives and records a stream of audio data from an RTP endpoint, and in parallel, decodes the data from its network format (for example, G.711A or G.723.1). The application also performs silence detection, DTMF detection, or both with the data.
Native record without inband silence and DTMF detection	Application receives and records a stream of audio data from an RTP endpoint. No data decoding takes place, inband silence detection is not supported, and DTMF detection is supported only through Fusion RFC 2833 support.

## Native play

To implement native play functionality, the application performs the following tasks:

- Opens the ADI API on the context and starts the NOCC protocol.
- Opens the MSPP API on the context and creates an RTP endpoint.
- Retrieves the filter ID of the RTP endpoint.
- Supplies the ADI API with information about the RTP audio streams and specifies the desired behavior for native play operations.
- Starts and stops playing audio data from a native audio stream.

The following illustration shows an overview of the native play mechanism:



## Sample procedure

When implementing native play functionality, applications use functions from the following resources:

- NaturalAccess functions to set up event queues and contexts, and to open services on the contexts.
- ADI API functions to start a protocol, to set native play settings, and to play audio data.
- MSPP API functions to create an RTP endpoint and retrieve the unique filter ID for the endpoint.

The following procedure shows the function sequence used to implement a typical native play operation:

Step	Action
1	<b>ctaCreateQueue</b> creates a NaturalAccess event queue. <code>ctaCreateQueue ( &amp;queuehd )</code>
2	<b>ctaCreateContext</b> creates a NaturalAccess context for the audio channel. <code>ctaCreateContext ( queuehd, &amp;ctahd )</code>
3	<b>ctaOpenServices</b> opens the ADI and MSPP APIs on the context. <code>ctaOpenServices ( ctahd, svclist, nsvcs )</code>
4	<b>adiStartProtocol</b> starts the nocc protocol on the ADI port. <code>adiStartProtocol ( ctahd, "nocc", NULL, startparms )</code>
5	<b>mspCreateEndpoint</b> creates an audio MSPP API RTP endpoint and returns an MSPP API endpoint handle ( <i>ephd</i> ). <code>mspCreateEndpoint ( ctahd, mspaddrstruct, mspparmstruct, &amp;rtpephd )</code>
6	<b>mspGetFilterHandle</b> retrieves the runtime filter ID ( <i>fltID</i> ) associated with the RTP endpoint handle ( <i>ephd</i> ). The application uses the returned <i>fltID</i> as the destination for the audio stream played out from the ADI port. <code>mspGetFilterHandle ( rtpephd, MSP_ENDPOINT_RTPEFDX, &amp;fltID )</code>
7	<b>adiSetNativeInfo</b> specifies both the context handle of the ADI port and the RTP endpoint <i>fltID</i> returned by <b>mspGetFilterHandle</b> , sets NMS native play parameters. <code>adiSetNativeInfo ( ctahd, NULL, fltID, fltID_parms )</code>
8	<b>adiPlayFromMemory</b> starts playing a message. <code>adiPlayFromMemory ( ctahd, encoding, buffer, bufsize, parms )</code>
9	<b>adiStopPlaying</b> stops playing the message. <code>adiStopPlaying ( ctahd )</code>

## Example

The following example shows how to perform a native play operation:

```
ret = ctaCreateQueue( NULL, 0, &hCtaQueueHd );

ret = ctaCreateContext( hCtaQueueHd, 0, "Play", &ctahd );

ServiceCount = 2;
ServDesc[0].name.svcname      = "ADI";
ServDesc[0].name.svcmgrname   = "ADIMGR";
ServDesc[0].mvipaddr.board    = board;
ServDesc[0].mvipaddr.mode     = 0;
ServDesc[1].name.svcname      = "MSP";
ServDesc[1].name.svcmgrname   = "MSPMGR";
ret = ctaOpenServices( ctahd, ServDesc, ServiceCount );
ret = WaitForSpecificEvent( CTAEVN_OPEN_SERVICES_DONE, &event );

ret = adiStartProtocol( ctahd, "nocc", NULL, NULL );
ret = WaitForSpecificEvent( ADIEVN_STARTPROTOCOL_DONE, &event );

// create mspp RTP endpoint
ret = mspCreateEndpoint( ctahd, &mspAddr, &mspParm, &ephd );
ret = WaitForSpecificEvent( MSPEVN_CREATE_ENDPOINT_DONE, &event );

// get cg6xxx board handle
ret = mspGetFilterHandle( msphd, MSP_FILTER RTPFDX_EPH, rtp_play_filter_handle );
ret = adiSetNativeInfo( ctahd, NULL, /* no ingress handle, as this is a play only */
    rtp_play_filter_handle, &natpr_ctl ); /* RTP endpoint filter ID
    specified as a destination for audio */
ret = adiPlayFromMemory( ctahd, ADI_ENCODE_EDTX_AMRNB, /* audio play */
    MemoryBuffer, RecordedBytes, NULL );
```

## Native record with inband silence and DTMF detection

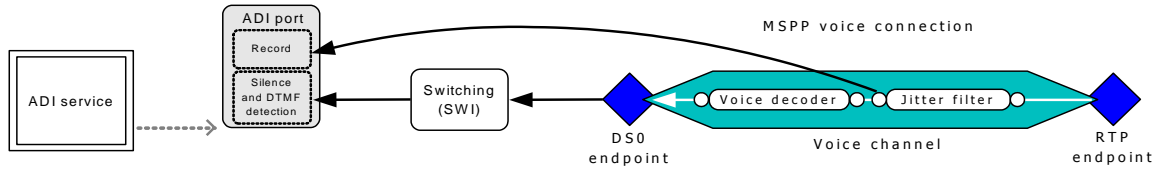
Use the following procedure to implement native record with inband silence detection (or DTMF detection without silence detection) for the CG boards.

To implement native record functionality with inband silence detection or DTMF detection, the application performs the following tasks:

- Opens the ADI API on a NaturalAccess context and starts the NOCC protocol on the context.
- Opens the MSPP API and the ADI API on a second context, and creates an RTP endpoint, a DS0 endpoint, and a voice channel on the context.
- Connects the RTP endpoint, DS0 endpoint, and voice channel to create a voice connection.
- Creates a switch connection between the ADI port and the DS0 endpoint.
- Retrieves the filter ID of the jitter filter associated with the voice channel.
- Supplies the ADI service with information about the RTP audio streams and specifies the desired behavior for native record operations.
- Starts and stops recording audio data from a network audio stream.



The following illustration shows an overview of the native record mechanism with voice decoding enabled:



### Sample procedure

Applications use functions from the following NaturalAccess resources to implement native record functionality with inband silence detection or DTMF detection:

- NaturalAccess functions to set up event queues and contexts and to open services on the contexts.
- ADI API functions to start a protocol, to set native record settings, and to record incoming audio data.
- MSPP functions to create a voice connection consisting of a voice decoding channel, an RTP endpoint, and a DS0 endpoint, and to retrieve the unique filter ID of the RTP endpoint's jitter filter.
- SWI functions to switch together the ADI service port and the MSPP service connection (through the DS0 endpoint).

The following procedure shows the function sequence used to implement a typical native record operation with decoding on CG boards:

Step	Action
1	<b>ctaCreateQueue</b> creates a NaturalAccess event queue. <code>ctaCreateQueue (&amp;queuehd)</code>
2	<b>ctaCreateContext</b> creates a NaturalAccess context for the audio channel. <code>ctaCreateContext (queuehd, &amp;ctahd)</code>
3	<b>ctaOpenServices</b> opens the ADI service on the context. When using <b>ctaOpenServices</b> , the application must specify the following: <ul style="list-style-type: none"> <li>• Set the <code>svclist.mvipaddr.mode</code> parameter to <code>ADI_VOICE_DUPLEX</code> to allocate DSP resources for the channel on the CG board.</li> <li>• Set the <code>svclist.mvipaddr.stream</code> parameter to 0 and the <code>svclist.mvipaddr.timeslot</code> parameter to a unique and valid entry. For more information, refer to the <i>Dialogic® NaturalAccess™ Software Developer's Manual</i>.</li> </ul> <code>ctaOpenServices (ctahd, svclist, nsvcs)</code>
4	<b>adiStartProtocol</b> starts the NOCC protocol on the audio channel and enables silence detection on the audio channel. <code>adiStartProtocol (ctahd, "nocc")</code>
5	<b>swiOpenSwitch</b> opens a switching device for the context and returns a switch handle ( <b>swihd</b> ). <code>swiOpenSwitch (ctahd, "cg6ksw", board, 0x0, &amp;swihd)</code>
6	<b>ctaCreateContext</b> creates a NaturalAccess context for the MSPP channel. <code>ctaCreateContext (queuehd, &amp;msphd)</code>
7	<b>ctaOpenServices</b> opens the MSPP API on the context. <code>ctaOpenServices (ctahd, svclist, nsvcs)</code>

Step	Action
8	<b>mSPCreateEndpoint</b> creates an audio DS0 endpoint and returns an endpoint handle ( <b>ephhd</b> ). <code>mSPCreateEndpoint (ctahd, mspaddrstruct, mspparmstruct, &amp;ds0ephhd)</code>
9	<b>mSPCreateChannel</b> creates a full duplex or voice decoding channel. <code>mSPCreateChannel (ctahd, chnladdr, chnlparms, &amp;chanhd)</code>
10	<b>mSPCreateEndpoint</b> creates an audio RTP endpoint and returns an endpoint handle. <code>mSPCreateEndpoint (ctahd, mspaddrstruct, mspparmstruct, &amp;rtppephd)</code>
11	<b>mSPConnect</b> connects the RTP and DS0 endpoints with the voice channel. <code>mSPConnect (rtpephd, chanhd, ds0ephhd)</code>
12	<b>swiMakeConnection</b> , with the <b>swihd</b> returned by <b>swiOpenSwitch</b> , connects the MSPP DS0 output to the ADI audio channel input and vice versa. When using <b>swiMakeConnection</b> , the application specifies the stream and timeslot used to create the ADI port and the stream and timeslot used to create the DS0 endpoint. <code>swiMakeConnection (swihd, fusion_ds0, adi_ds0, 1)</code>
13	<b>mSPGetFilterHandle</b> retrieves the filter identifier ( <b>fltID</b> ) associated with the MSPP record channel. <code>mSPGetFilterHandle (chanhd, MSP_FILTER_JITTER, &amp;fltID)</code>
14	<b>adiSetNativeInfo</b> , with both the context handle of the ADI port and the <b>fltID</b> returned by <b>mSPGetFilterHandle</b> , sets NMS native record parameters. <code>adiSetNativeInfo (ctahd, fltID, NULL, natpr_parms)</code>
15	<b>adiRecordToMemory</b> starts recording a message. <code>adiRecordToMemory (ctahd, buf, bufsize, rec_param)</code>
16	<b>adiStopRecording</b> stops recording the audio portion of the message. <code>adiStopRecording (ctahd)</code>

## Example

The following example shows how to perform a native record operation that supports ADI silence and DTMF detection on CG boards:

```
ret = ctaCreateQueue( NULL, 0, &hCtaQueueHd );
ret = ctaCreateContext( hCtaQueueHd, 0, "Record", &ctahd );

ServiceCount = 2;
ServDesc[0].name.svcname      = "ADI";
ServDesc[0].name.svcmgrname   = "ADIMGR";
ServDesc[0].mvipaddr.mode     = ADI_VOICE_DUPLEX;
ServDesc[0].mvipaddr.stream   = 0;
ServDesc[0].mvipaddr.timeslot = record_timeslot;
ServDesc[1].name.svcname      = "MSP";
ServDesc[1].name.svcmgrname   = "MSPMGR";

ret = ctaOpenServices( ctahd, ServDesc, ServiceCount );
ret = WaitForSpecificEvent( CTAEVN_OPEN_SERVICES_DONE, &Event );

// IP Channel Initialization
MSPHD    ds0_ephhd;
MSPHD    rtp_ephhd;

// Create and init RTP endpoint
MSP_ENDPOINT_ADDR    rtpaddr    = {0};
MSP_ENDPOINT_PARAMETER rtp_params = {0};

rtpaddr.size      = sizeof(MSP_ENDPOINT_ADDR);
rtpaddr.eEpType   = MSP_ENDPOINT_RTPFDX;
rtpaddr.nBoard    = g_Board;
```

```

...
mspCreateEndpoint( ctaHd, &rtpaddr, &rtp_params, &rtp_ephd );
if (! WaitForSpecificEvent(MSPEVN_CREATE_ENDPOINT_DONE, &Event, 5000))
{
    printf("Failed waiting for MSPEVN_CREATE_ENDPOINT_DONE (RTP)");
    return FAILURE;
}

// create mspp DS0 endpoint
MSP_ENDPOINT_ADDR    ds0addr    = {0};
ds0addr.eEpType       = MSP_ENDPOINT_DS0;
ds0addr.nBoard        = board;
ds0addr.size          = sizeof(MSP_ENDPOINT_DS0);
ds0addr.EP.DS0.nTimeslot = record_timeslot;
MSP_ENDPOINT_PARAMETER ds0parms  = {0};
ds0parms.size         = sizeof(DS0_ENDPOINT_PARMS);
ds0parms.eParmType    = MSP_ENDPOINT_DS0;
ds0parms.EP.DS0.media = MSP_VOICE;
mspCreateEndpoint( ctaHd, &ds0addr, &ds0parms, &ds0_ephd );
if (! WaitForSpecificEvent(MSPEVN_CREATE_ENDPOINT_DONE, &Event, 5000))
{
    printf("Failed waiting for MSPEVN_CREATE_ENDPOINT_DONE (DS0)");
    return FAILURE;
}

// create mspp Channel
MSP_CHANNEL_ADDR    chanaddr    = {0};
MSP_CHANNEL_PARAMETER chan_params = {0};

chanaddr.nBoard      = Board;
chanaddr.channelType = G711FullDuplex;
chanaddr.FilterAttribs = MSP_FCN_ATTRIB_RFC2833;
chan_params.size      = sizeof( MSP_CHANNEL_PARAMETER );
chan_params.channelType = G711FullDuplex;
chan_params.ChannelParms.VoiceParms.size = sizeof( MSP_VOICE_CHANNEL_PARMS );
...
// Create channel
mspCreateChannel( ctaHd, &chanaddr, &chan_params, &msphd );
CTA_EVENT CtaEvent;
if (! WaitForSpecificEvent(MSPEVN_CREATE_CHANNEL_DONE, &Event, 5000))
{
    printf("Failed waiting for MSPEVN_CREATE_CHANNEL_DONE");
    return FAILURE;
}

// connect mspp endpoints
ret = mspConnect(rtp_ephd, msphd, ds0_ephd);
if (! WaitForSpecificEvent(MSPEVN_CONNECT_DONE, &Event, 5000))
{
    printf("Failed waiting for MSPEVN_CONNECT_DONE");
    return FAILURE;
}

// enable channel
mspEnableChannel(msphd);
if (! WaitForSpecificEvent(MSPEVN_ENABLE_CHANNEL_DONE, &Event, 5000))
{
    printf("Failed waiting for MSPEVN_ENABLE_CHANNEL_DONE");
    return FAILURE;
}

//adiStartProtocol
adiStartProtocol( ctahd, "nocc", NULL, NULL );
if (! WaitForSpecificEvent( ADIEVN_STARTPROTOCOL_DONE, &Event, 5000))
{
    printf("Failed to receive ADIEVN_STARTPROTOCOL_DONE event");
    return FAILURE;
}

// get cg6xxx board handle
ret = mspGetFilterHandle( msphd, MSP_FILTER_JITTER, &cg6xxx_board_filter_handle );

```

```

ADI_NATIVE_CONTROL parms = {0};      /* Native parameters */
parms.frameFormat      = 0;
parms.include2833      = 0;
parms.vadFlag          = 0;
parms.nsPayload        = 0;
parms.mode             = ADI_NATIVE;
parms.rec_encoding     = ADI_ENCODE_EDTX_MU_LAW;
parms.payloadID        = 0;
ret = adiSetNativeInfo( ctahd, cg6xxx_board_filter_handle,
    NULL, /* this is record only so no egress handle */
    &parms );

// get default adi record parms
ret = ctaGetParms( ctahd, ADI_RECORD_PARMID, &recparms, sizeof(ADI_RECORD_PARMS) );
ret = adiRecordToMemory( ctahd, ADI_ENCODE_EDTX_MU_LAW, /* audio rec */
    MemoryBuffer, RecordedBytes, &recparms );

```

## Native record without inband silence and DTMF detection

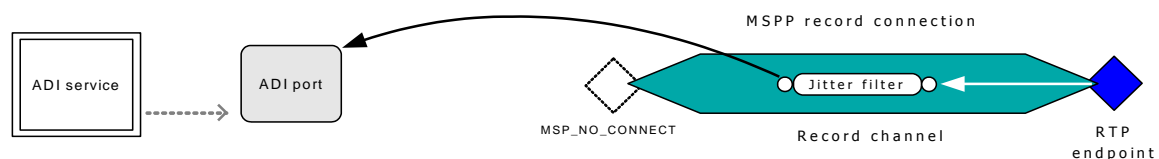
To implement native record functionality without inband silence detection or DTMF detection, the application performs the following tasks:

- Creates a NaturalAccess event queue and context.
- Opens the ADI API and the MSPP API on the context and starts the NOCC protocol on the context.
- Creates an MSPP RTP endpoint and an MSPP record channel on the context.
- Connects the RTP endpoint with the record channel to create a record connection.
- Retrieves the filter ID of the jitter filter within the record channel.
- Supplies the ADI API with information about the RTP audio streams and specifies the desired behavior for native record operations.
- Starts and stops recording audio data from a network audio stream.

The application must zero the novoicetime, silencetime, and beeptime record parameters, to prevent starting the silence detector and tone generator.

**Note:** Applications can perform DTMF detection using Fusion RFC 2833 support, but silence detection is not supported. For more information, refer to the RFC 2833 support chapter in this *Dialogic® NaturalAccess™ Fusion™ VoIP API Developer's Manual*.

The following illustration shows an overview of the native record mechanism without voice decoding:



## Sample procedure

Applications use functions from the following NaturalAccess resources to implement native record functionality without inband silence detection or DTMF detection:

- NaturalAccess functions to set up event queues and contexts, and to open services on the contexts.
- ADI API functions to start a protocol, to set native record settings, and to record incoming audio data.
- MSPP functions to create a voice connection consisting of a record channel and an RTP endpoint, and to retrieve the unique filter ID associated with the record channel.

The following procedure shows function sequence used to implement a typical native record operation without decoding:

Step	Action
1	<b>ctaCreateQueue</b> creates a NaturalAccess event queue. <code>ctaCreateQueue ( &amp;queuehd )</code>
2	<b>ctaCreateContext</b> creates a NaturalAccess context for the audio channel. <code>ctaCreateContext ( queuehd, &amp;ctahd )</code>
3	<b>ctaOpenServices</b> opens the ADI and MSPP services on the context. <code>ctaOpenServices ( ctahd, svclist, nsvcs )</code>
4	<b>adiStartProtocol</b> starts the nocc protocol on the audio channel. <code>adiStartProtocol ( ctahd, "nocc", NULL, startparms )</code>
5	<b>mspCreateChannel</b> creates a record channel. <code>mspCreateChannel ( ctahd, chnladdr, chnlparms, &amp;chanhd )</code>
6	<b>mspCreateEndpoint</b> creates an audio RTP endpoint and returns an endpoint handle ( <b>ephd</b> ). <code>mspCreateEndpoint ( ctahd, mspaddrstruct, mspparmstruct, &amp;rtpephd )</code>
7	<b>mspConnect</b> , with MSP_NO_CONNECT instead of a DS0 endpoint handle, connects the record channel with the RTP endpoint. <code>mspConnect ( rtpephd, chanhd, MSP_NO_CONNECT )</code>
8	<b>mspGetFilterHandle</b> retrieves the filter identifier ( <b>fltID</b> ) associated with the MSPP record channel. <code>mspGetFilterHandle ( chanhd, MSP_FILTER_JITTER, &amp;fltID )</code>
9	<b>adiSetNativeInfo</b> , using both the context handle of the ADI port and the <b>fltID</b> returned by <b>mspGetFilterHandle</b> , sets NMS native record parameters. <code>adiSetNativeInfo ( ctahd, fltID, NULL, natpr_parms )</code>
10	<b>adiRecordToMemory</b> starts recording audio data. <code>adiRecordToMemory ( ctahd, buf, bufsize, rec_param )</code>
11	<b>adiStopRecording</b> stops recording audio data. <code>adiStopRecording ( ctahd )</code>

## Example

The following example shows how to perform a native record operation without decoding:

```
ret = ctaCreateQueue( NULL, 0, &hCtaQueueHd );
ret = ctaCreateContext( hCtaQueueHd, 0, "Record", &ctahd );

ServiceCount = 2;
ServDesc[0].name.svcname      = "ADI";
ServDesc[0].name.svcmgrname   = "ADIMGR";
ServDesc[0].mvipaddr.mode     = ADI_VOICE_DUPLEX;
ServDesc[0].mvipaddr.stream   = 0;
ServDesc[0].mvipaddr.timeslot = record_timeslot;
ServDesc[1].name.svcname      = "MSP";
ServDesc[1].name.svcmgrname   = "MSPMGR";

ret = ctaOpenServices( ctahd, ServDesc, ServiceCount );
ret = WaitForSpecificEvent( CTAEVN_OPEN_SERVICES_DONE, &Event );

// IP Channel Initialization
MSPHD      ds0_ephd = MSP_NO_CONNECT;
MSPHD      rtp_ephd;

// Create and init RTP endpoint
...
mspCreateEndpoint( ctaHd, &rtpaddr, &rtp_params, &rtp_ephd );
if ( ! WaitForSpecificEvent( MSPEVN_CREATE_ENDPOINT_DONE, &Event, 5000 ) )
{
    printf("Failed waiting for MSPEVN_CREATE_ENDPOINT_DONE (RTP)");
    return FAILURE;
}

chanaddr.nBoard      = Board;
chanaddr.channelType = G711RecordChannel;
chanaddr.FilterAttribs = MSP_FCN_ATTRIB_RFC2833;
chan_params.size      = sizeof( MSP_CHANNEL_PARAMETER );
chan_params.channelType = G711RecordChannel;
chan_params.ChannelParms.VoiceParms.size = sizeof( MSP_VOICE_CHANNEL_PARMS );

// Create channel
mspCreateChannel( ctaHd, &chanaddr, &chan_params, &msphd );
CTA_EVENT CtaEvent;
if ( ! WaitForSpecificEvent( MSPEVN_CREATE_CHANNEL_DONE, &Event, 5000 ) )
{
    printf("Failed waiting for MSPEVN_CREATE_CHANNEL_DONE");
    return FAILURE;
}

// connect mspp endpoints
ret = mspConnect( rtp_ephd, msphd, MSP_NO_CONNECT );
if ( ! WaitForSpecificEvent( MSPEVN_CONNECT_DONE, &Event, 5000 ) )
{
    printf("Failed waiting for MSPEVN_CONNECT_DONE");
    return FAILURE;
}

// enable channel
mspEnableChannel( msphd );
if ( ! WaitForSpecificEvent( MSPEVN_ENABLE_CHANNEL_DONE, &Event, 5000 ) )
{
    printf("Failed waiting for MSPEVN_ENABLE_CHANNEL_DONE");
    return FAILURE;
}

//adiStartProtocol
adiStartProtocol( ctahd, "nocc", NULL, NULL );
if ( ! WaitForSpecificEvent( ADIEVN_STARTPROTOCOL_DONE, &Event, 5000 ) )
{
    printf("Failed to receive ADIEVN_STARTPROTOCOL_DONE event");
    return FAILURE;
}
```

```
}

// get cg6xxx board handle
ret = mSPGetFilterHandle( msphd, MSP_FILTER_JITTER, &cg6xxx_board_filter_handle );

ADI_NATIVE_CONTROL parms = {0};      /* Native parameters */
parms.frameFormat          = 0;
parms.include2833         = 0;
parms.vadFlag             = 0;
parms.nsPayload           = 0;
parms.mode                = ADI_NATIVE;
parms.rec_encoding        = ADI_ENCODE_EDTX_MU_LAW;
parms.payloadID           = 0;
ret = adiSetNativeInfo( ctahd, cg6xxx_board_filter_handle,
    NULL, /* this is record only so no egress handle */
    &parms);

// get default adi record parms
ADI_RECORD_PARMS recparms;
ret = ctaGetParms( ctahd, ADI_RECORD_PARMID, &recparms, sizeof(ADI_RECORD_PARMS) );
recparms.novoicetime = 0;
recparms.silencetime = 0;
recparms.beeptime = 0
ret = adiRecordToMemory( ctahd, ADI_ENCODE_EDTX_MU_LAW, /* audio rec */
    MemoryBuffer, RecordedBytes, &recparms );
```





---

# 13 Implementing DCE over IP transport

---

## Fusion DCE transport

---

DCE transport over IP transmits fax or modem traffic over a clear Fusion G.711 channel. Applications use the *echo\_v4.f54* echo cancellation module provided with Natural Access, along with the Fusion RFC 2833 compliant capabilities, to implement this functionality.

Fax machines and modems generate one of the ANS (2100 Hz) family of signals as part of initial negotiation. The *echo\_v4.f54* module detects the variations of this signal, and sends notifications to the host to initiate the transition to a G.711 clear channel.

The remote gateway is informed of the occurrence of an ANS family signal when it receives an RFC 2833 packet containing a specific ANS signal identification. When it receives this packet, the remote gateway transitions into a G.711 clear channel and configures the echo canceller appropriately.

The application transitions from Fusion voice channel to a Fusion G.711 clear channel under the following circumstances:

- Emitting side: A DS0 endpoint detects an ANS-type signal
- Receiving side: An RTP endpoint receives an ANS-type RFC 2833 packet

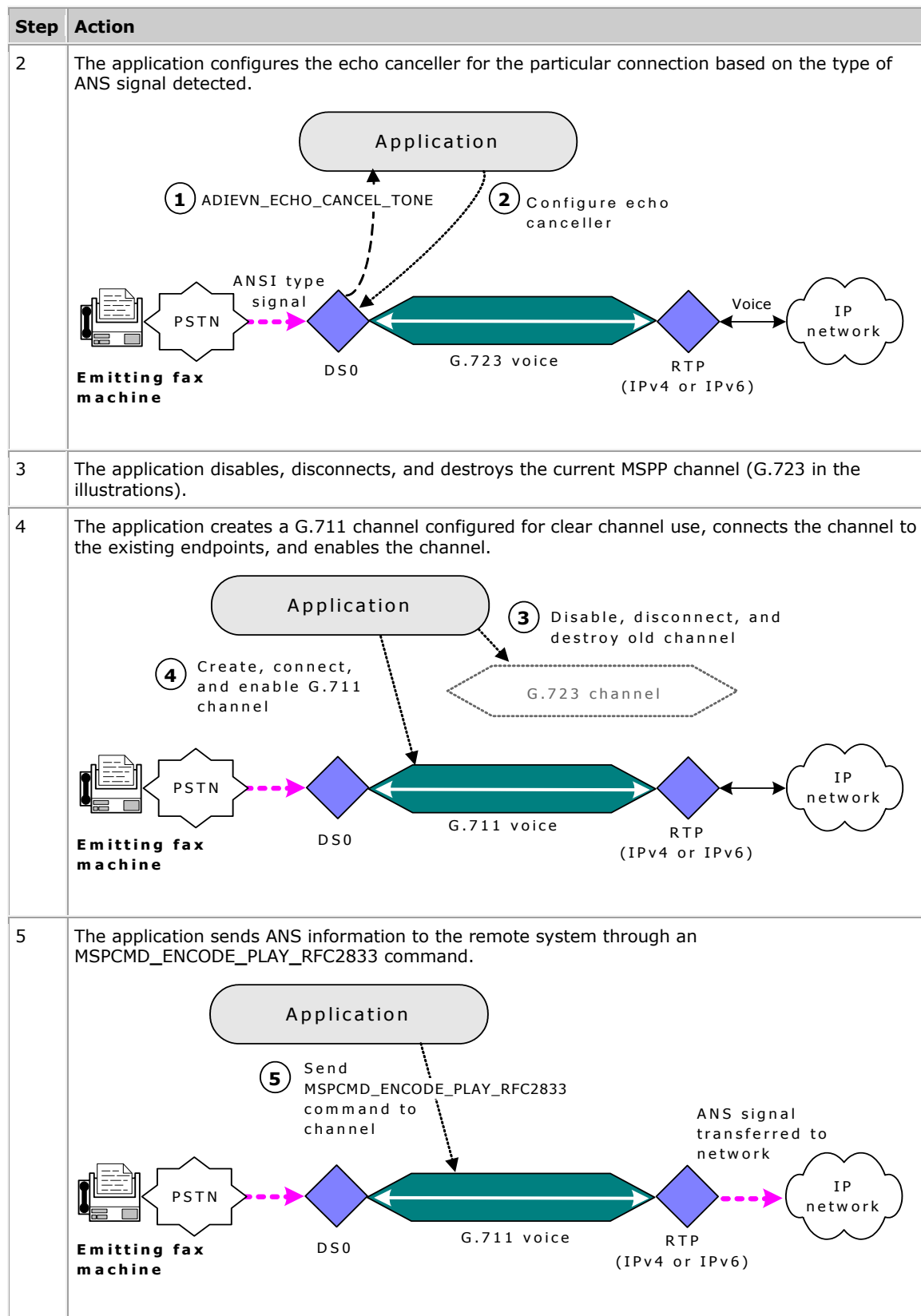
This topic shows the required transition steps on the emitting and receiving side of the IP network.

### Emitting side

---

The transition from voice data to DCE transport over IP occurs in the following way on the emitting side of the network:

Step	Action
1	The MSPP DS0 endpoint (through its echo canceller functionality) detects an ANS-type signal from the PSTN network it and sends an ADIEVN_ECHO_CANCEL_TONE event to the application.



For more information, refer to *Transferring non-DTMF RFC 2833 events* on page 77.

## Receiving side

The transition from voice data to DCE transport over IP occurs in the following way on the emitting side of the network:

Step	Action
1	RTP endpoint detects an ANS-type RFC 2833 packet and sends MSPEVN_RFC2833_REPORT to the application.
2	The application configures the echo canceller based on the type of ANS signal detected. <div data-bbox="324 483 1364 861"> <p>Diagram illustrating Step 2: The application configures the echo canceller. The diagram shows an IP network connected to an RTP endpoint (IPv4 or IPv6). An ANSI type signal is sent from the RTP endpoint to the Application. The Application sends an MSPEVN_RFC2833_REPORT back to the RTP endpoint and configures an echo canceller. The RTP endpoint sends G.723 voice data to a DS0 endpoint, which then sends Voice data to a PSTN and a Receiving fax machine.</p> </div>
3	The application disables, disconnects, and destroys the current MSPP channel.
4	The application creates a G.711 channel configured for clear channel use, connects the channel to the existing endpoints, and enables the G.711 channel. <div data-bbox="324 1029 1364 1407"> <p>Diagram illustrating Step 4: The application creates a G.711 channel. The diagram shows the previous G.723 channel being disabled and destroyed. The Application sends a signal to the RTP endpoint to disable the old channel and create a new G.711 channel. The RTP endpoint sends G.711 voice data to a DS0 endpoint, which then sends Voice data to a PSTN and a Receiving fax machine.</p> </div> <p>DCE fax data can now be transferred through the two gateways across the IP network.</p>

## Setting up DCE transport capability

To implement DCE transport in Fusion, applications must perform the following tasks:

Step	Action
1	<p>Configure the board resources by specifying the <i>echo_v4</i> module in the Resource keyword string in the appropriate CG board keyword file.</p> <p>The following example shows how to specify a Fusion channel with DCE transport capability in a CG board configuration resource definition string.</p> <pre>Resource[0].Definitions = (echo_v4.ln20_ap25 &amp; \     ((f_g723.cod_rfc2833 &amp; f_g723.dec_rfc2833)   \     (f_g729a.cod_rfc2833 &amp; f_g729a.dec_rfc2833)   \     (f_g711.cod_rfc2833 &amp; f_g711.dec_rfc2833)))</pre>
2	<p>Start a protocol on the context associated with the voice channel, and specify appropriate mediamask parameters to enable echo cancellation, but <i>not</i> to detect CED and other signals within the DS0 endpoint. For more information about configuring echo cancellation appropriately for DCE over IP transport refer to <i>Configuring the echo canceller</i> on page 101.</p> <p>The following sample code shows how an application defines a flag that specifies the appropriate mediamask settings to support DCE transport.</p> <pre>if (pCmdLineParms-&gt;bEnableDCETransport) {     startparms.mediamask =     NCC_CC_AUTOECHO; } else {     startparms.mediamask =     NCC_CC_RESVDTMF       NCC_CC_RESVSILENCE       NCC_CC_RESVCLRDWN       NCC_CC_AUTODTMF       NCC_CC_AUTOECHO; }</pre>
3	<p>When creating the voice channel, specify settings in the MSPP G.711 channel address and parameter structures that enable RFC 2833 capabilities. For more information about setting appropriate G.711 voice channel parameters to implement DCE over IP transport, refer to <i>Configuring clear G.711 channels</i> on page 102.</p>

## Configuring the echo canceller

To implement DCE transport, configure the echo canceller so that the echo suppressor is disabled, or so that the echo canceller is disabled and operates in bypass mode.

The following table shows the actions the application must take depending on the type of signal detected:

Signal	Frequency	Amplitude modulation	Phase reversals	Action to take on both ends
ANS	2100 +/- 15 Hz	0	0	Disable echo suppressor
ANS/	2100 +/- 15 Hz	0	450 +/- 25 ms	Disable echo canceller and echo suppressor
ANSam	2100 +/- 1 Hz	15 +/- 0.1 Hz	0	Disable echo suppressor
ANSam/	2100 +/- 1 Hz	15 +/- 0.1 Hz	450 +/- 25 ms	Disable echo canceller and echo suppressor

The following sample code shows how to configure the echo canceller with echo suppression disabled and to disable the echo canceller so that it operates in bypass mode:

```

If (nEchoSuppress)
{
EchoParms[chn].size = sizeof(ADI_ECHOCANCEL_PARMS);
EchoParms[chn].mode |= ADI_ECHOCANCEL_SUPPRESS;

ret = adiModifyEchoCanceller(hPstnCallCtl[chn], &EchoParms[chn]);
}
else if (nEchoBypass)
{
EchoParms[chn].size = sizeof(ADI_ECHOCANCEL_PARMS);
EchoParms[chn].mode |= ADI_ECHOCANCEL_BYPASS;

ret = adiModifyEchoCanceller(hPstnCallCtl[chn], &EchoParms[chn]);
}

```

## Configuring clear G.711 channels

---

When creating G.711 channels to implement DCE over IP transport, the application must configure the channel so that it is free from distortion to minimize the risk of poor playout by the decoder. The application accomplishes this by specifying the following G.711 channel settings:

- Configure the G.711 voice channel jitter filter for static operation, and set the jitter depth large enough to allow a sufficient flow of frames to the decoder filter within the channel.
- If the application sets channel parameters at create time, it must set the encoder parameter NotchControl to zero, which disables the notch filter. However, the default encoder parameters for a G.711 encoder filters do support DCE transport.

The following example shows the proper jitter configuration for DCE transport:

```
ChannelParms.VoiceParms.JitterParms.size = sizeof( msp_FILTER_JITTER_PARMS );  
ChannelParms.VoiceParms.JitterParms.depth = MSP_CONST_JITTER_DEPTH_MAX;  
ChannelParms.VoiceParms.JitterParms.adapt_enabled = 0;
```

---

# 14 Using T.38 fax connections

---

## T.38 fax connections overview

---

The ITU T.38 standard describes a mechanism for transferring facsimile documents in real-time between standard Group 3 facsimile terminals or Internet Aware Facsimile (IAF) devices over the Internet or other networks using IP protocols. It also specifies a protocol by which IP fax gateways or IAF devices exchange messages and data over an IP network.

This topic describes:

- T.38 fax transmission model
- T.38 fax relay
- Fusion T.38 fax channels

**Note:** Fusion does not currently support IPv6 for T.38 fax connections.

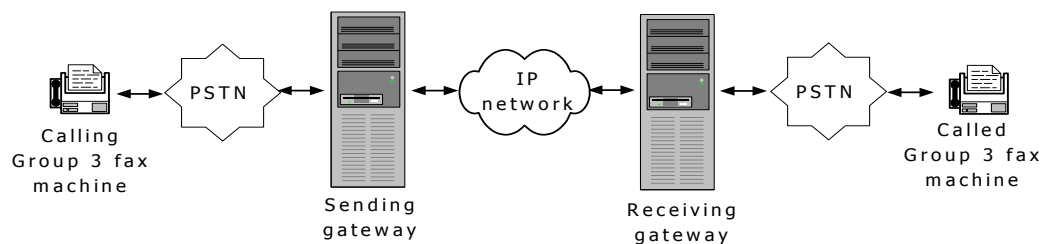
## T.38 fax transmission model

---

The process of transferring facsimile documents by IP fax gateways includes the following steps:

- Demodulating incoming T.30 fax signals at the sending or emitting gateway.
- Translating T.30 fax signals into T.38 Internet fax protocol (IFP) packets.
- Exchanging IFP packets between emitting and receiving T.38 gateways.
- Translating T.38 IFP packets back into T.30 signals at the receiving gateway
- Modulating T.30 signals and transferring them to the receiving fax machine and vice versa.

Gateways that support T.38 fax are usually transparent to the fax machine user. The following illustration shows an overview of the basic model for Fusion T.38 fax transmission:



## T.38 fax relay

Fax relay transmissions occur in real-time. A calling fax machine places a PSTN call and is connected through a PSTN line to a gateway system. The emitting gateway demodulates the fax data and then relays the data to another receiving gateway (in packet form).

When it receives the IP packets, the receiving gateway recovers the fax data, re-modulates the data, and transfers the data to the called fax machine over a PSTN line. Under normal conditions, the packet network connection is transparent to the fax machines and they operate as if they were connected directly over a PSTN. Although T.38 fax relay offers the advantage of real-time fax transmission, it is sensitive to network parameters such as network latency, jitter, and packet loss.

## T.38 fax standard

The ITU-T T.38 standard describes a fax relay oriented protocol and specifies the messages exchanged between gateways while transferring fax data over IP networks. The following items are specified by this standard:

- The order of messages transported across the packet network
- The message format required
- Error correction techniques
- IP call setup mechanisms (in the Annexes section)

The ITU-T T.38 standard specifies how data is transmitted from one gateway to the another. It does not specify how the sending gateway demodulates fax data from the calling fax machine or how fax data is re-modulated to the called fax machine.

## Fusion T.38 fax channels

Fusion provides the following two types of T.38 fax components:

Channel type	Description
T.38 fax full duplex channel	General purpose full duplex T.38 fax channel that transfers T.38 data between a DS0 and T38UDP endpoint. The channel also demodulates or re-modulates the fax data stream depending on the direction (towards the IP or PSTN interface) that the data is moving.
T38UDP full duplex endpoint	Provides an entry point and an exit point for T.38 fax data transferred through the system. Also performs processing tasks such as UDPTL encoding and decoding, and IFP encoding and decoding with the data that flows through the endpoint.  When the T38UDP endpoint is enabled, it listens for T.38 fax packets on the incoming data stream. When it detects the first valid T.38 packet, it returns an MSPEVN_T38_PACKET_DETECTED unsolicited event to the application. Disabling and then re-enabling the T38UDP endpoint resets the endpoint.



## Using T.38 fax full duplex channels

---

There are several ways to implement T.38 fax on Fusion systems so that the application can switch from voice transmission to fax transmission during the course of a phone call.

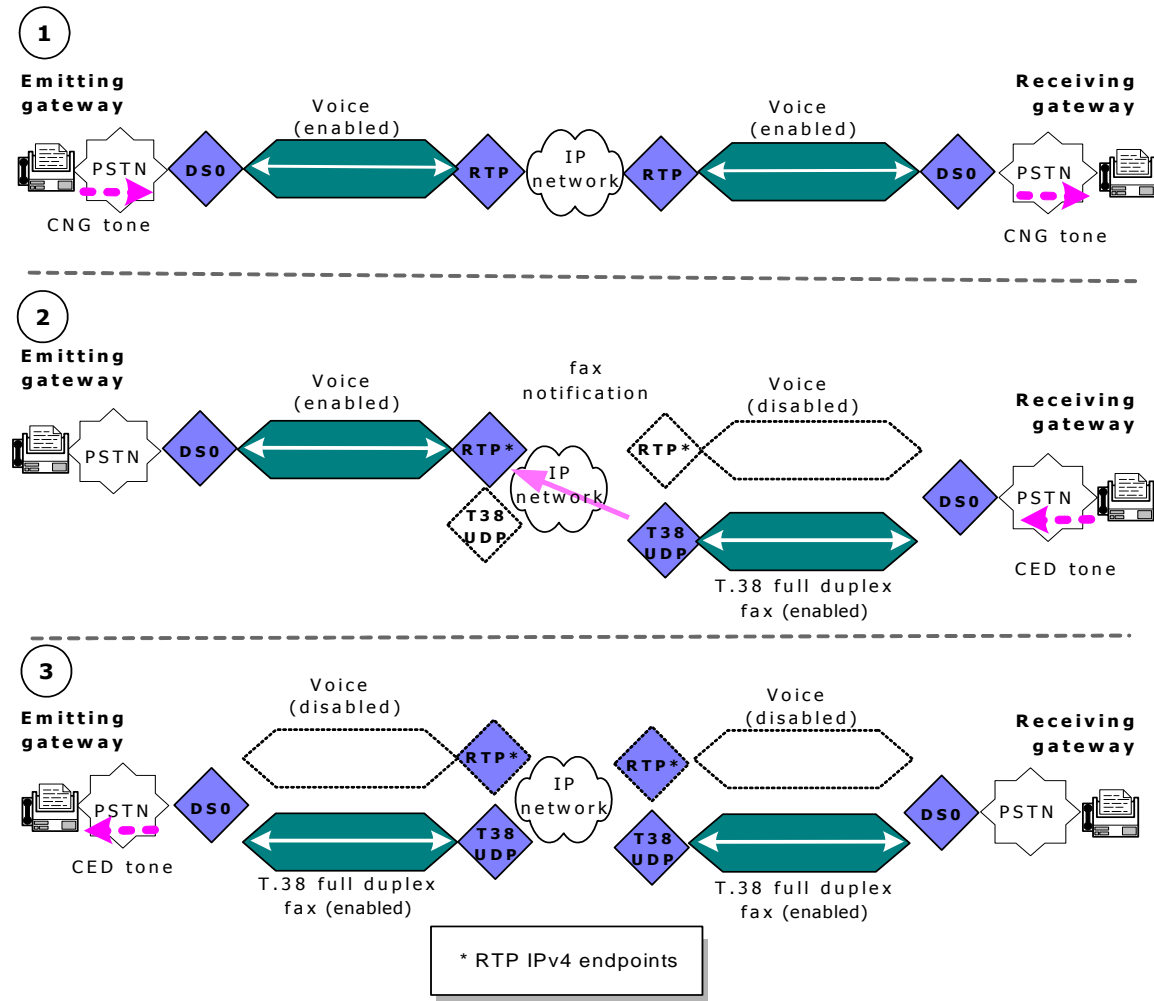
One way to implement this switch is to create consecutive MSPP voice encoding and decoding connections and T.38 fax full duplex channels on Fusion systems for transferring data across the IP network. When the application switches from voice data transmission to fax data transmission (either based on some external call control indication or the detection of a CED tone through the ADI service), it first tears down the voice channel and then sets up a fax channel to transfer the fax data.

When the receiving gateway detects the start of a T.30 fax transmission (for example, detects a CED tone via ADI service tone detection):

Step	Action
1	The receiving gateway disables the voice connection. It then creates and enables a T.38 fax full duplex connection using the same DS0 endpoint, and a new T38UDP endpoint and a new T.38 fax channel. The application also sends a MSPP filter command to the DS0 endpoint that switches the endpoint's transmission mode from voice to fax.
2	The receiving gateway notifies the emitting gateway (through a reliable transport medium such as TCP or a call control protocol such as SIP, MEGACO or H.248) that fax data is forthcoming.
3	The emitting gateway performs the same tasks as the receiving gateway to switch from voice to fax transmission, and can begin the fax transmission.

When the application disables the voice channel and creates a new T.38 fax full duplex connection, it can assign the UDP port number formerly used by the voice channel's RTP IPv4 endpoint to the new T38UDP endpoint.

The configuration shown in the following illustration conserves UDP ports by using the same UDP port for the voice connections and the fax connections (active RTP IPv4 and T38UDP endpoints cannot use the same UDP ports at the same time). However, this configuration requires considerable application interaction, not only to set up and tear down MSPP channels, but also to send fax data notification messages across the network at the appropriate times.



## Setting up fax and voice connections

---

Fusion gateway applications can implement T.38 fax transmission by switching from voice to fax transmission over existing connections. Applications perform a set of tasks to create voice connections and T.38 fax full duplex connections in advance, and then switch from voice to fax transmission when fax activity is detected. This process is called voice/fax switchover.

To set up voice/fax switchover, the application creates the following MSPP components for each connection:

- DS0 endpoint
- RTP IPv4 endpoint
- T38UDP endpoint
- Voice channel
- T.38 fax full duplex channel

Since the MSPP service allows a single endpoint to connect to multiple channels, applications use **mppConnect** to connect the DS0 endpoint to:

- A voice channel (and an RTP IPv4 endpoint) creating a voice connection.
- A T.38 fax full duplex channel (and a T38UDP endpoint) creating a T.38 fax stand-by connection.

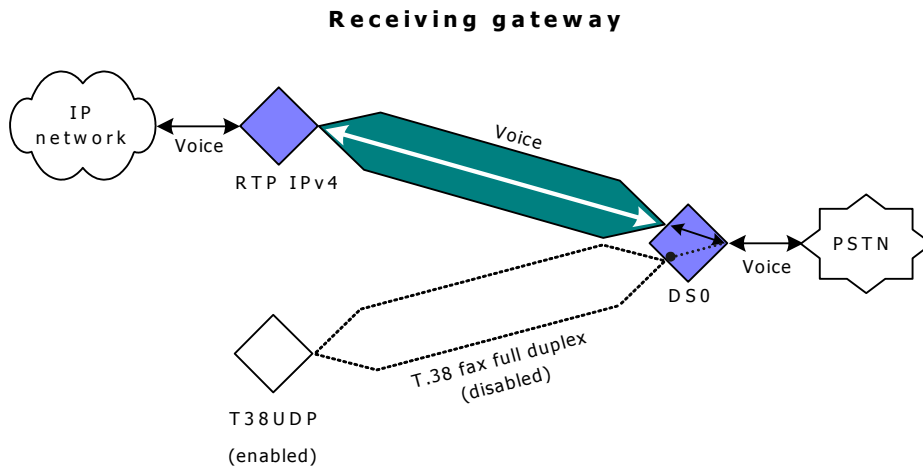
In T.38 fax stand-by configurations, the application must assign different UDP port numbers to RTP IPv4 and T38UDP endpoints. The T38UDP endpoint must be enabled before it can to perform packet detection.

By default, voice/fax switchover connections transfer voice data until the gateway application directs otherwise. When the application is alerted of fax activity, it switches from voice data transmission to fax data transmission.

Fax activity detection is performed by one of the following MSPP components, depending on the side of the fax transmission where the gateway resides:

- Receiving side: The application performs switchover when the ADI service detects a CED tone (through the DS0 endpoint).
- Emitting side: The application performs switchover when the T38UDP endpoint detects a valid T.38 packet and sends an MSPEVN\_T38\_PACKET\_DETECTED unsolicited event.

The following illustration shows a gateway configured to perform voice/fax switchover:

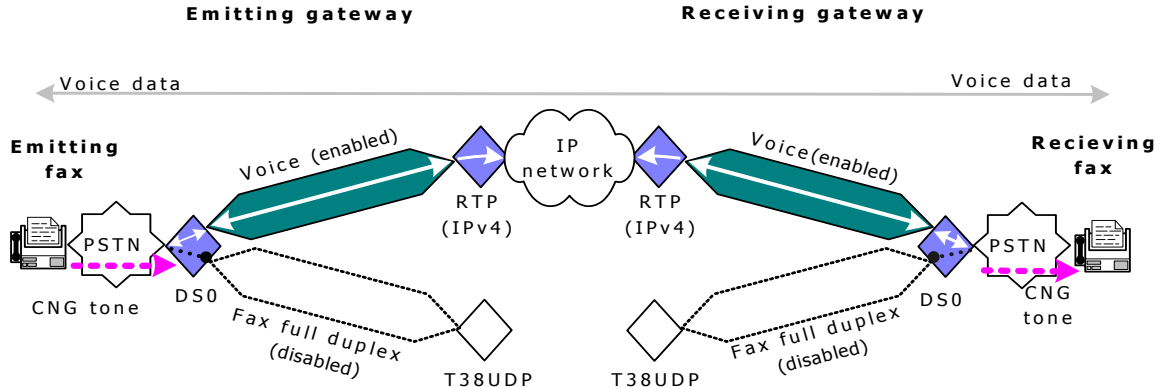


### Connection setup tasks

The gateway application must perform the following tasks to set up a voice/fax switchover connection:

Step	Action
1	Create a DS0 endpoint (with ADI service tone detection enabled), an RTP IPv4 endpoint, and a T38UDP endpoint (using <b>mspCreateEndpoint</b> ).
2	Create a full duplex voice channel and a T.38 fax full duplex (using <b>mspCreateChannel</b> ).
3	Connect (using <b>mspConnect</b> ) the DS0 endpoint to both: <ul style="list-style-type: none"> <li>A voice channel (and an RTP IPv4 endpoint), creating a voice connection.</li> <li>A T.38 fax full duplex channel (and an T38UDP endpoint), creating a T.38 fax stand-by connection.</li> </ul> Connecting the DS0 endpoint in this way creates two full duplex MSPP connections that share the same DS0 endpoint.
4	Enable the voice channel (with <b>mspEnableChannel</b> ).
5	Enable ADI service (CED) tone detection (with <b>adiStartToneDetector</b> ).

To implement voice/fax switchover gateways at different locations on the network, you must set up voice/fax connections at both network locations. The following illustration shows two Fusion gateways sending voice data across an IP network through voice/fax switchover connections. Initially, the connections on both ends of the network are set up to transfer voice data.



When a calling fax machine sends a CNG tone (indicating that a fax terminal is calling), the tone is carried across the network as voice data. The called fax machine responds by sending a CED tone.

## Performing voice and fax switchover

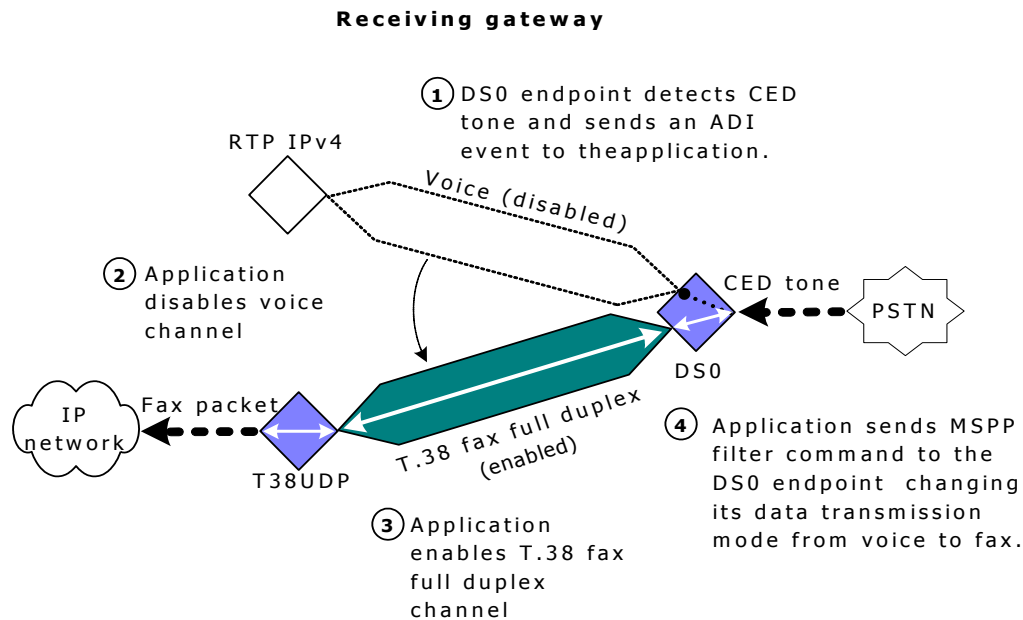
The procedure for implementing voice/fax switchover varies depending whether the gateway is at the emitting side or receiving side of the fax transmission.

### Receiving gateway

When the called fax machine receives the CNG tone, it responds by sending a CED tone, which indicates that the called station is a fax terminal. Voice/fax switchover takes place in the following way:

Step	Action
1	The DS0 endpoint in the receiving gateway detects the CED tone (through ADI tone detection) and sends an ADI event to the application.
2	The application disables the voice channel.
3	The application waits for the MSPEVN_DISABLE_CHANNEL_DONE event from the voice channel, then enables the T.38 fax full duplex channel.
4	The application sends a MSP_CMD_DS0_CONFIG filter command to the DS0 endpoint, changing the endpoint's data transmission mode from voice to fax.

The fax CED tone is transferred as a T.38 packet to the IP network. The following illustration shows the receiving gateway performing voice/fax switchover:



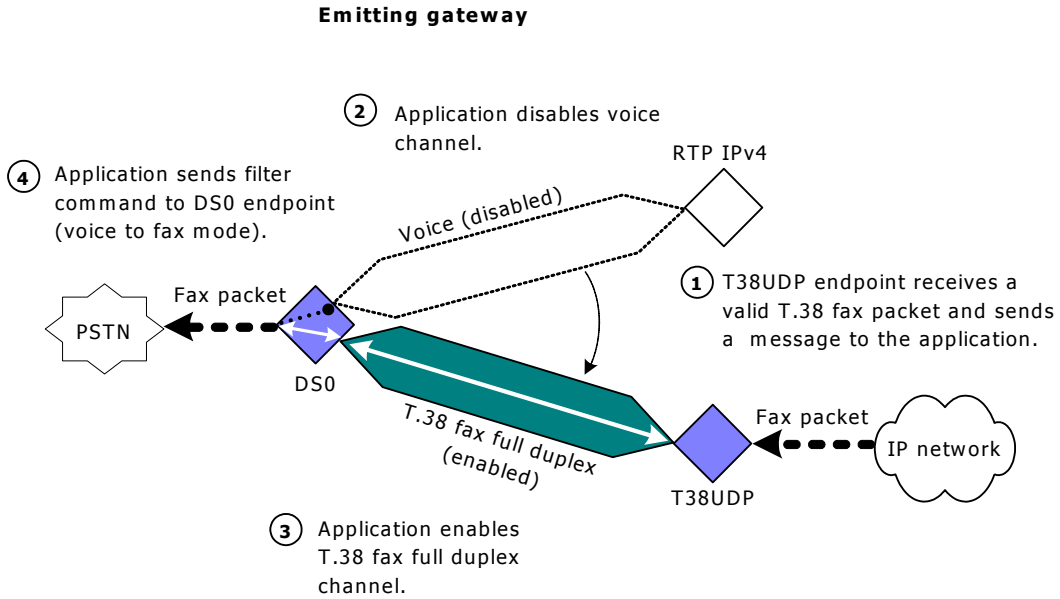
### Emitting gateway

On the emitting gateway, voice/fax switchover takes place in the following way:

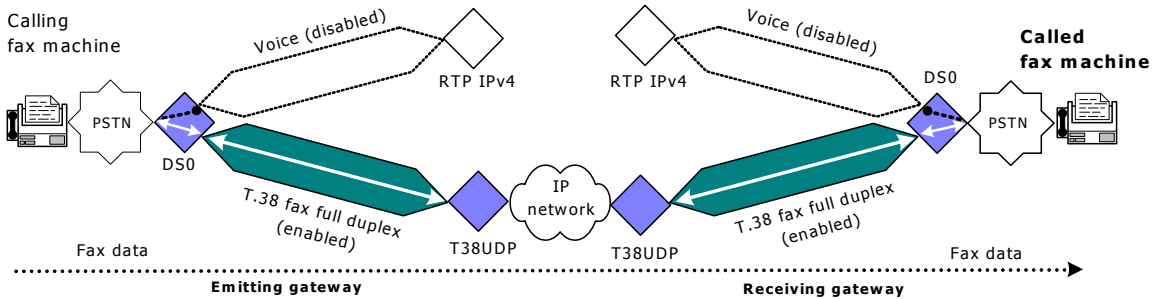
Step	Action
1	When the gateway receives the first T.38 packet, the T38UDP endpoint automatically sends an MSPEVN_T38_PACKET_DETECTED unsolicited event to the gateway application.
2	The emitting gateway disables the voice channel.
3	The application enables the T.38 fax full duplex channel.
4	The application sends an endpoint filter command that switches the DS0 endpoint from voice to fax transmission mode. The T.38 full duplex channel regenerates the CED tone and sends it over the PSTN network (through the DS0 endpoint).

When the calling fax machine (connected to the emitting gateway) receives the CED tone, it begins to transmit fax data across the network. The T38UDP endpoint must be enabled to detect the arrival of T.38 fax packets.

The following illustration shows the emitting gateway performing voice/fax switchover:





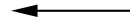













The following illustration shows the emitting and receiving gateways transferring fax data after voice/fax switchover has taken place:



## Switching from voice to fax

Fusion gateways configured for voice/fax switchover initially treat all calls as voice transmissions. The following table shows how Fusion gateways switch from voice to fax processing mode in the presence of fax data:

Step	Fax station 1 (calling machine)	Emitting gateway	Receiving gateway	Fax station 2 (called machine)
1	Sends CNG tone 	 Transfers CNG tone over a voice connection.	 Transfers CNG tone over a voice connection.	 Receives the CNG tone Sends a CED tone 
2	 Receives CED tone	 Receives the T.38 packet: <ul style="list-style-type: none"> <li>T38UDP endpoint detects the first valid T.38 packet and sends an event to the application.</li> <li>Application disables the voice channel and enables the fax channel.</li> <li>Application sends a command to DS0 endpoint, switching it from voice to fax transmission mode.</li> <li>Fax channel transfers fax data to PSTN.</li> </ul>	 Receives the CED tone: <ul style="list-style-type: none"> <li>DS0 endpoint detects the CED tone and sends an event to the application.</li> <li>Application disables the voice channel and enables the fax channel.</li> <li>Application sends a command to DS0 endpoint, switching it from voice to fax transmission mode.</li> <li>Fax channel starts sending T.38 packets.</li> </ul>	
3	 Fax transmission	 Fax transmission	 Fax transmission	 Fax transmission
4	 Disconnects	 Ends fax transmission (or disconnects) and switches DS0 endpoint back to voice mode.	 Disconnects and switches DS0 endpoint back to voice mode.	 Disconnects

After the fax connection is disabled, you can reuse the connection after disabling and then re-enabling the T38UDP endpoint.

For information about MSPP service functions and information about MSPP service T.38 filters and filter commands, refer to the *Dialogic® NaturalAccess™ Media Stream Protocol Processing API Developer's Manual*.



## Receiving fax billing events

---

T.38 fax full duplex channels consist of a single T.38 fax relay channel filter. T.38 fax relay filters can return billing events that show the status of T.38 fax transmissions. By default, T.38 fax relay filters do not return any unsolicited events. To enable this feature, the application uses **mspSendCommand** to send an **MSP\_CMD\_FAXRELAY\_CONFIG** command to the appropriate T.38 fax relay filter. An **msp\_FILTER\_FAXRELAY\_CONFIG** structure included with the command specifies an **eventmask** parameter. This parameter determines the types of unsolicited events the T.38 fax relay filter generates.

Applications can specify the following unsolicited event masks:

Mask	Events
FAXRELAY_EVENTMASK_REPT_FAX_BEGIN_END	Reports the beginning and ending of a fax session.
FAXRELAY_EVENTMASK_REPT_PASSED_PAGE	Reports successful passing of a fax page.
FAXRELAY_EVENTMASK_DEFAULT	Returns to the default filter configuration. No fax session events were returned.

Applications must invoke **mspReleaseBuffer** to release the buffers returned with T.38 fax unsolicited events.

For more information about structures returned with T.38 fax unsolicited events, refer to the *mspunsol.h* header file. For more information about sending commands to MSPP endpoint and channel filters, refer to the *Dialogic® NaturalAccess™ Media Stream Protocol Processing API Developer's Manual*.



# 15 Using ThroughPacket multiplexing

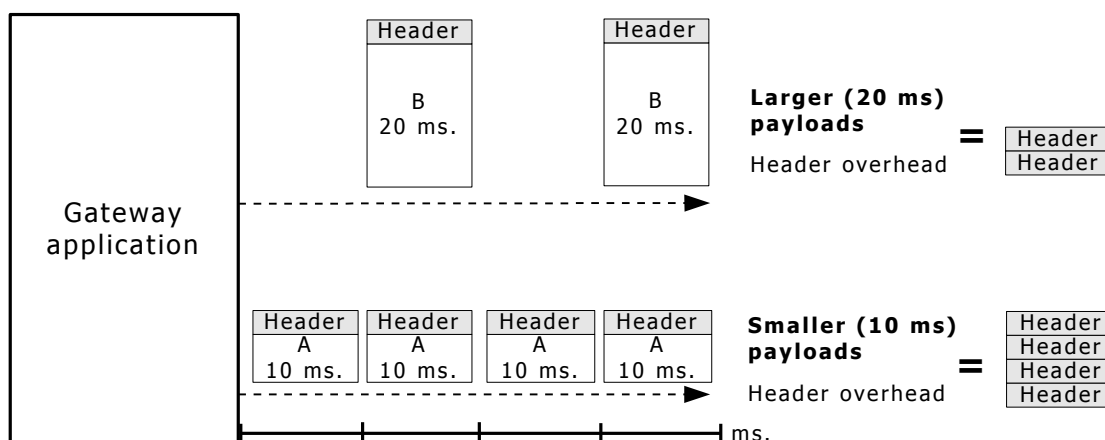
## ThroughPacket multiplexing overview

One way to decrease audio latency between the speaker and the listener in packet-based voice transmissions is to send smaller payloads of voice data. This reduces audible delay, but increases the number of packets that need to be transferred for a given voice data transmission. Since each of these smaller audio payloads requires a set amount of header information, the overall effect is to increase the total amount of data that the gateway must transmit.

Typically, voice data is transmitted over a packet-switched network like the Internet in a continuous stream of discrete RTP data packets transferred on a per-channel basis. The header portion of the packet represents overhead associated with the actual voice data. For information transmitted across IP connections, the packet header includes a link header, an IP header, a UDP header, and an RTP header. This overhead is fixed in size, even if the packet payload size decreases.

As gateways systems reduce packet payload size in an effort to decrease per-session latency, packet headers represent an increasing percentage of the total packet size. Transmitting smaller packets also means transmitting more packets (and therefore a greater number of headers) in order to transfer the same amount of data. An increase in the number of packets that gateways transmit can also impose a cumulative increase in the workload for network routers and, overall, contribute to a reduction of quality in network transmissions.

The following illustration shows how the proportion of network bandwidth used to transmit packet headers (or overhead) increases as the size of packet payloads decreases:

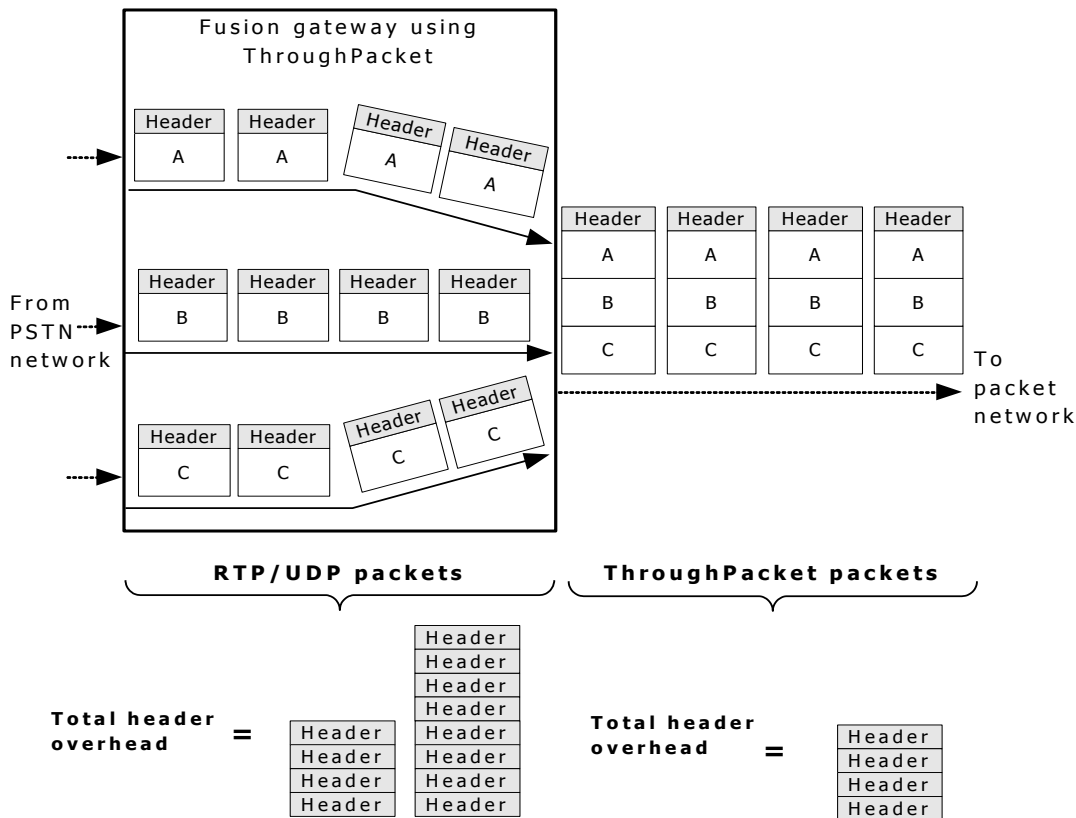


**Note:** Fusion does not support IPv6 for ThroughPacket fax connections.

## ThroughPacket (TPKT) endpoints

ThroughPacket is a proprietary multiplexing algorithm for grouping together payloads from multiple voice streams directed to the same destination (for example, a remote gateway). ThroughPacket (TPKT) endpoints provide a mechanism by which gateways systems group voice data payloads from separate voice channels into larger packets. ThroughPacket headers contain any information the receiving gateway needs to de-multiplex the packets and send the de-multiplexed payloads to their appropriate destinations.

Each TPKT packet carries the combined payloads of several separate voice sessions, yet contains only a single header. Therefore, the application sends fewer total packets, while devoting a smaller portion of its data output to transferring header information. The following illustration shows a Fusion gateway using the ThroughPacket multiplexing method to send payloads from multiple sessions in common packets:



ThroughPacket's enhanced efficiency in transferring data reduces the total bandwidth the application needs to transfer voice data across the network. By multiplexing many streams of RTP data, a VoIP system employing ThroughPacket technology is able to provide two major benefits:

- Reduced data rate
- Reduced packet rate

## Increasing data rate efficiency

When VoIP systems use ThroughPacket instead of RTP to transfer voice data, they substantially reduce the packet header overhead associated with individual data packets. This reduced packet size also leads to a reduced data rate that helps to minimize the costs associated with maintaining a wide area network (WAN). WAN facility costs are generally associated with the network's aggregate data rate, as well as the distance over which systems transfer a given volume of data. Therefore, gateway systems engineered with ThroughPacket can substantially reduce the costs associated with long distance IP-based data transfer.

The following table shows data rate reduction that ThroughPacket can achieve in a gateway system. The TPKT packet data rate efficiency factor column provides a measure of how effectively ThroughPacket reduces bandwidth from a data rate perspective when the system uses different standard VoIP vocoders. As the table shows, gateways reduce bandwidth most significantly when they use low bit rate vocoders.

Vocoder type	Vocoder rate (kbit/s)	Default payload (ms)	Packet rate for 120 RTP streams (packets/sec)	Packet rate for equivalent TPKT stream (packets/sec)	TPKT packet data rate efficiency factor
G.711	64.0	20	9,600.00	8640.00	1.11
G.726	32.0	20	5,760.00	4,320.00	1.33
G.723.1	6.40	30	2048.00	940.41	2.18
G.723.1	5.33	30	1920.00	808.42	2.37
G.729a	8.00	20	2,880.00	1,212.63	2.38

## Reducing packet rates

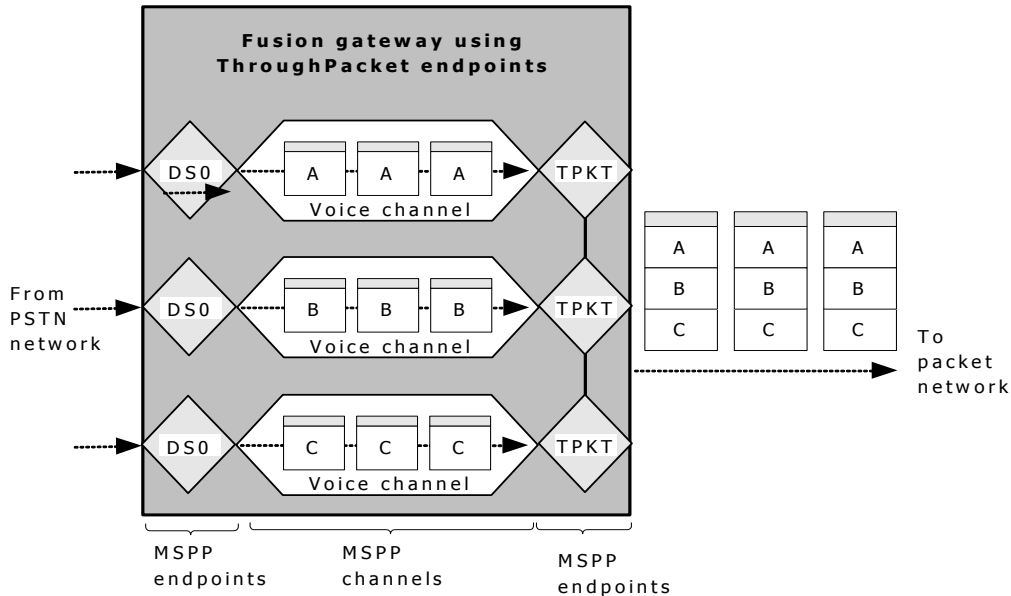
ThroughPacket is essentially a trunking technique. It concatenates the data from multiple channels into a single packet. This technique significantly reduces the number of packets that the gateway needs to transmit each second. By sending fewer packets through the network, ThroughPacket reduces the load on the network routers. Since routing performance (for example, jitter and dropped packets) tends to decrease with increased packet rate, ThroughPacket can reduce routing costs and improve voice quality. Because packet traffic in IP networks tends to take place in bursts, reducing the packet rate from any source can reduce the chances of packet loss due to router congestion at peak periods.

The following table compares the packet rate for 120 simplex RTP streams versus a single ThroughPacket stream that multiplexes the 120 voice streams. The TPKT packet data rate efficiency factor column shows results for different standard vocoders. The greatest benefits occur when using the low bit rate vocoders.

Vocoder type	Vocoder rate (kbit/s)	Default payload (ms)	Packet rate for 120 RTP streams (packets/sec)	Packet rate for equivalent TPKT stream (packets/sec)	TPKT packet data rate efficiency factor
G.711	64.00	20	6,000.00	750.00	8.00
G.726	32.00	20	6,000.00	375.00	16.00
G.723.1	6.40	30	4,000.00	81.63	49.00
G.723.1	5.33	30	4,000.00	70.18	57.00
G.729A	8.00	20	6,000.00	105.26	57.00

## Implementing ThroughPacket multiplexing

Fusion applications can use MSPP service ThroughPacket (TPKT) endpoints to reduce the total bandwidth needed to transfer multiple sessions of data to a common destination over a packet network. The following illustration shows a Fusion gateway using MSPP service TPKT endpoints to multiplex the payloads from several voice channels into larger packets:



When Fusion endpoints at the destination gateway receive TPKT packets, they de-multiplex the data and direct the individual payloads to their separate destinations.

To implement TPKT endpoints on a Fusion system, you must:

1. Configure TPKT parameters in the CG board's keyword file (using the OAM configuration parameters) and boot the board.
2. Create and configure TPKT endpoints with MSPP service functions.

Applications can create MSPP TPKT endpoints in one of two modes.

Mode	Description
Complex	TPKT transport where payloads from different sessions are combined into larger packets to reduce the number of packets transferred to the network.
Simple	TPKT transport where separate media sessions are carried in separate packets. This type of transport supports proprietary Clarent systems and does not reduce the bandwidth required by the application. Use RTP transport for individual VIP sessions.

## Using ThroughPacket OAM board keywords

---

To use TPKT endpoints on a particular CG board, you must specify appropriate parameters in the board's OAM API keyword file. These parameters specify the following:

- TPKT DLM file to load to the CG board.
- Data transmission parameters for TPKT endpoints created on the board.

After setting the TPKT and DLM file keywords, boot the board with an OAM utility such as *oamsys* for the parameters to take effect. For more information about *oamsys*, refer to the *Dialogic® NaturalAccess™ OAM System Developer's Manual*.

This topic describes TPKT keywords and provides an example of ThroughPacket keywords implemented in an OAM API board keyword file.

### ThroughPacket resource (DLM) file

---

CG boards that support TPKT endpoints need to include a downloadable module (DLM) file for performing ThroughPacket processing. Load the TPKT DLM file (*cg6ktpkt.dlm*) by referencing the TPKT DLM in the CG board's keyword file.

The following three files must be referenced (in a DLMFile string) in the CG board keyword file for boards that use ThroughPacket:

```
#####  
# DOWNLOADABLE RUNTIME MODULES  
DLMFile[0] = cg6krun  
DLMFile[1] = cg6kFusion  
DLMFile[2] = cg6ktpkt  
#####
```



## TPKT data transmission parameters

In addition to a TPKT DLM file reference, boards that support TPKT endpoints must include TPKT data transmission parameters in their board keyword files. The following TPKT keywords define the way the CG board transfers data from TPKT endpoints:

Keyword	Minimum	Maximum	Description
TPKT.Enable	NA	NA	Enables (value = 1) or disables (value = 0) ThroughPacket support.
TPKT.SimpleRxPort	1024	65535	UDP port number to receive simple packets.
TPKT.SimpleTxPort	1024	65535	UDP port number to transmit simple packets.
TPKT.ComplexRxPort	1024	65535	UDP Port to receive complex packets.
TPKT.ComplexTxPort	1024	65535	UDP Port to transmit complex packets.
TPKT.NumberOfComplexForwardConditions	1	8	Number of conditions set for ThroughPacket transmission.
TPKT.ComplexForward.Count	8	8	Number of ThroughPacket transmission conditions defined for the system. Always set this keyword to 8 (that is, you must always define eight conditions, although some conditions can be set to NULL).
TPKT.ComplexForward[0].LifeTimeTicks	0	99	Number of timer ticks (in 10 ms increments) before a ThroughPacket packet can be sent out.
TPKT.ComplexForward[0].DestinationPacketSize	1	1440	Threshold value indicating the number of bytes that must be accumulated in the packet payload before the packet can be sent out.

For more information about OAM ThroughPacket keywords, refer to the CG board manual.

## Example

---

The following example shows keywords used to configure TPKT endpoint data transmission parameters for a CG board used by a Fusion gateway:

```
TPKT.Enable = 1
TPKT.SimplexRxPort = 49152
TPKT.SimplexTxPort = 49152
TPKT.ComplexRxPort = 49153
TPKT.ComplexTxPort = 49153
TPKT.NumberOfComplexForwardConditions = 4
TPKT.ComplexForward.Count = 8
TPKT.ComplexForward[0].LifeTimeTicks = 0
TPKT.ComplexForward[0].DestinationPacketSize = 1440
TPKT.ComplexForward[1].LifeTimeTicks = 1
TPKT.ComplexForward[1].DestinationPacketSize = 980
TPKT.ComplexForward[2].LifeTimeTicks = 2
TPKT.ComplexForward[2].DestinationPacketSize = 700
TPKT.ComplexForward[3].LifeTimeTicks = 3
TPKT.ComplexForward[3].DestinationPacketSize = 1
TPKT.ComplexForward[4].LifeTimeTicks = 0
TPKT.ComplexForward[4].DestinationPacketSize = 0
TPKT.ComplexForward[5].LifeTimeTicks = 0
TPKT.ComplexForward[5].DestinationPacketSize = 0
TPKT.ComplexForward[6].LifeTimeTicks = 0
TPKT.ComplexForward[6].DestinationPacketSize = 0
TPKT.ComplexForward[7].LifeTimeTicks = 0
TPKT.ComplexForward[7].DestinationPacketSize = 0
```

If none of the `TPKT.ComplexForward.LifeTimeTicks` keywords is set to 0, the default maximum payload size is automatically set to 1440.

## Creating TPKT endpoints

The MSPP service provides a standard set of endpoints that are each capable of sending and receiving data to and from a particular network-specific format. These endpoints provide sources through which data can enter and leave the system. MSPP TPKT endpoints perform payload multiplexing and de-multiplexing with the voice data that is transferred through them.

Applications create MSPP endpoints by invoking the MSPP service function **mspCreateEndpoint**. To create a TPKT endpoint, the application specifies a *ctahd* associated with a MSPP service instance and specifies TPKT parameters in two structures:

Structure	Parameters
TPKT endpoint address structure	<pre>typedef struct tag_MSP_ENDPOINT_ADDR {     DWORD nBoard; // board number     DWORD eParmType;     MSP_ENDPOINT_TPKT } MSP_ENDPOINT_ADDR;</pre>
TPKT endpoint parameter structure	<pre>typedef struct tag_MSP_ENDPOINT_PARMS {     DWORD size;     MSP_ENDPOINT_PARAMETER     DWORD eParmType;     MSP_ENDPOINT_DS0,     union     {         typedef struct {             DWORD localSessionID;             DWORD localSessionSeq;             DWORD remoteSessionID;             DWORD remoteSessionSeq;             DWORD deliveryMethod;             BYTE remoteGatewayIP[4];             DWORD coderType;         } msp_TPKT_ENDPOINT_CONFIG;     } EP; } MSP_ENDPOINT_PARAMETER;</pre>

UDP port conflicts can occur if the application running on the host system uses UDP port number already associated with an RTP or TPKT endpoint or vice versa. To guarantee that no UDP port conflicts occur when opening a UDP socket on the board that will be used for a TPKT session, the application can open a socket with the same UDP port on the host. Opening a socket in this way causes the host IP stack to generate unique UDP port numbers for each RTP or TPKT session without creating address conflicts. For more information, refer to the CG board manual.

## Specifying TPKT endpoint parameters

This topic provides the following information:

- TPKT endpoint parameters
- Session IDs
- Session sequence flags

### TPKT endpoint parameters

`mSP_TPKT_ENDPOINT_CONFIG` parameters specify the following endpoint information:

Name	Allowed values	Description
<code>localSessionID</code>	0 - 4094	Local session ID.
<code>localSessionSeq</code>	0   1	Sequence number for detecting session overlap.
<code>remoteSessionID</code>	0 - 4094	Remote session ID.
<code>remoteSessionSeq</code>	0   1	Sequence number for detecting session overlap.
<code>deliveryMethod</code>	<code>kComplexPacket</code> : complex packets <code>kSimplePacket</code> : simple packets	Session delivery method (simple or complex).
<code>remoteGatewayIP</code>	Valid IP address. This field contains an array of four bytes (0 - 255 each) for specifying the remote gateway IP address.	Destination gateway IP address for the endpoint.
<code>coderType</code>	The following coder types are available: <ul style="list-style-type: none"> <li>• <code>kG723High</code>: G.723.1 6.4 kbit/s</li> <li>• <code>kG723Low</code>: G.723.1 5.3 kbit/s</li> <li>• <code>kG729</code>: G.729A at 8.0 kbit/s</li> <li>• <code>kG711Alaw_64</code>: G.711 A law at 64 kbit/s</li> <li>• <code>kG711Mulaw</code>: G.711 mu law at 64 kbit/s</li> <li>• <code>kG726_32</code>: G.726 ADPCM at 32 kbit/s</li> </ul>	TPKT endpoint vocoder type.

### Session IDs

When the application creates TPKT endpoints, the `localSessionID` parameter assigns a session identifier to the endpoint. This session ID (an integer in the range of 0 to 4094) remains associated with the endpoint until the application destroys the endpoint. The application uses the `remoteSessionID` parameter to specify the ID of the session on the remote destination gateway.

Gateways exchange `remoteSessionID` and `remoteSessionSeq` information while performing IP call control prior to initiating TPKT sessions. Applications can change TPKT endpoint session IDs by using **`mSPSendCommand`** and the `MSP_CMD_TPKT_CONFIG` command. For more information, refer to *Sending commands to TPKT endpoints* on page 127.

## Session sequence flags

---

The SessionSeq parameters are flags with a value of 0 or 1. Applications use these parameters to detect session overlap. Session overlap occurs when an application disables a TPKT endpoint and then re-enables the endpoint to send data to a different session. Applications use localSessionSeq and remoteSessionSeq parameters to avoid these overlaps.

For example, when the application creates an active connection using a TPKT endpoint, the application can set the TPKT endpoint's localSessionSeq parameter to zero. If the application disables and then re-enables the TPKT endpoint, it can use **mspSendCommand** to reset the localSessionSeq parameter to 1. The remote TPKT endpoint uses the session sequence flag to discard packets intended for the disabled session. During the transition period when the remote gateway is not yet aware that the session has been disabled, the MSPP service running on the local gateway detects any packets directed to the disabled session and drops them.

For more information about creating MSPP endpoints, refer to the *Dialogic® NaturalAccess™ Media Stream Protocol Processing API Developer's Manual*.

## Connecting TPKT endpoints

Applications use **mSPConnect** to join TPKT endpoints with MSPP channels to create end-to-end gateway media paths. TPKT endpoints can connect with the following MSPP service channel types:

Channel type	Payload data rate	Basic frame duration	VAD support	Recommended frames/packet
G.711	64 kbit/s	10 ms	N/A	1
G.723.1	6.4 kbit/s or 5.33 kbit/s	30 ms	G.723.1 Annex A	1
G.729A	8 kbit/s	10 ms	G.729 Annex B	1
G.726	32 kbit/s	10 ms	N/A	1

T.38 fax channels are not currently supported for TPKT endpoints.

When maintaining 120 channels of voice data, using a number of frames per packet lower than the minimum indicated (for the specific vocoder) in the preceding table can exceed the CG board's processing capabilities. To reduce latency, use one frame per packet for all vocoders.

For more information about connecting MSPP endpoints and channels, refer to the *MSPP Service Developer's Reference Manual*.

## Configuring G.723.1 vocoders for use with TPKT endpoints

MSPP endpoints vocoders use static payload IDs as defined by RFC 1890. TPKT endpoints map to different payload IDs. The following table provides a mapping of TPKT vocoders to NMS vocoders:

Vocoder Description	TPKT Mapping	NMS Vocoder	NMS Payload ID
G.723.1 at 6.4 kbit/s	1	G723	4
G.723.1 at 5.33 kbit/s	2	G723	(4) 127 *

As the table shows, the default payload ID the MSPP service associates with G.723.1 vocoders (4) assumes operation at 6.4 kbit/s. Applications can dynamically change the TPKT endpoint's payload ID to a value associated with 5.33 kbit/s operation (127) by invoking the MSPP service function **mSPSendCommand**.

TPKT endpoints distinguish between G.723.1 vocoders running at 6.4 kbit/s and G.723.1 vocoders running at 5.33 kbit/s. Standard Fusion G.723.1 vocoders, in accordance with RFC1890, do not distinguish between these vocoder types. You must change the payload ID for G.723.1 voice encoders and decoders to 127 when configuring the G.723.1 channel for 5.33 kbit/s operation.

Applications perform the following steps to configure TPKT endpoints to use G.723.1 encoders or decoders for 5.33 kbit/s operation.

Step	Action
1	Set the coderType parameter in the TPKT endpoint msp_TPKT_ENDPOINT_CONFIG structure to kG723Low. If the application is creating the endpoint, it can specify this parameter in the endpoint's configuration parameters. If the endpoint is already created, the application must disable the endpoint first, and then send a command to the endpoint to change its coderType to kG723Low. For more information about sending commands to TPKT endpoints, refer to <i>Sending commands to TPKT endpoints</i> on page 127.
2	Send a command to the voice decoder filter to change the decoder payload ID to 127.
3	Send a command to the voice encoder filter to change encoder payload ID to 127.
4	Send a command to the voice encoder filter to change the encoder rate to 53.

For more information about sending commands to MSPP endpoints, refer to the *Dialogic® NaturalAccess™ Media Stream Protocol Processing API Developer's Manual*.

## Sending commands to TPKT endpoints

Applications can change settings specified in the msp\_TPKT\_ENDPOINT\_CONFIG structure by using **mspSendCommand**. When invoking **mspSendCommand** for TPKT endpoints, application specify an MSPP handle associated with a TPKT endpoint, the endpoint ID (the MSP\_ENDPOINT\_TPKT parameter for TPKT endpoints), and an endpoint command ID (MSP\_CMD\_TPKT\_CONFIG for TPKT endpoint commands). Applications then use the **mspBuildCommand** macro to concatenate the endpoint ID and command ID, while specifying the location of a buffer that includes the parameter values to set for the endpoint.

When an application wants to re-configure an TPKT endpoint, it must disable the endpoint before sending it new configuration information.

Applications specify the TPKT parameter values in a msp\_TPKT\_ENDPOINT\_CONFIG structure that contains the following information:

Name	Description
localSessionID	Local session ID.
localSessionSeq	Sequence number for detecting session overlap (0   1).
remoteSessionID	Remote session ID.
remoteSessionSeq	Sequence number for detecting session overlap (0   1).
deliveryMethod	Session delivery method (simple or complex).
remoteGatewayIP	Destination gateway for this session.
coderType	TPKT endpoint vocoder type.

For more information about sending commands to MSPP endpoints, refer to the *Dialogic® NaturalAccess™ Media Stream Protocol Processing API Developer's Manual*.

## Sending queries to TPKT endpoints

Applications can send queries to a TPKT endpoints by invoking the **mSPSendQuery** function. This function returns a buffer containing information about the specified endpoint. When invoking **mSPSendQuery** for TPKT endpoints, the application includes the MSPP handle associated with the endpoint, and concatenates the endpoint ID (MSP\_ENDPOINT\_TPKT for TPKT endpoints) and the endpoint query ID (MSP\_QRY\_TPKT\_ENDPOINT for TPKT endpoint queries) using the **mSPBuildQuery** macro.

MSP\_QRY\_TPKT\_CHANNEL queries return TPKT configuration information in a mSP\_TPKT\_ENDPOINT\_QUERY structure. This structure contains an mSP\_TPKT\_ENDPOINT\_ENTRY substructure and a mSP\_TPKT\_ENDPOINT\_STATS substructure.

The mSP\_TPKT\_ENDPOINT\_ENTRY substructure provides the following information:

Parameter	Description
channelState	State of the session: 0 - Created 1 - Configured 2 - Active
localSessionID	Local session ID.
remoteSessionID	Remote session ID.
localSessionSeq	Sequence number for detecting session overlap (0 or 1).
remoteSessionSeq	Sequence number for detecting session overlap (0 or 1).
deliveryMethod	Session delivery method (simple or complex).
remoteGatewayIP	Destination gateway for this session.
coderType	TPKT vocoder type.

The mSP\_TPKT\_ENDPOINT\_STATS substructure provides the following information:

Parameter	Type	Description
rxPkts	DWORD	Packets received from the network since the endpoint became active.
txPkts	DWORD	Packets sent to the network since the endpoint became active.

For information about sending queries to MSPP endpoints, refer to the *Dialogic® NaturalAccess™ Media Stream Protocol Processing API Developer's Manual*.



---

# 16 RTP MIB extended management component (EMC)

---

## RTP MIB EMC

---

Fusion enables applications to monitor information about RTP sessions in a chassis by using an OAM Extended Management Component (EMC) for RTP. The topics that follow provide a brief introduction to this EMC. For more information about OAM and its EMCs, refer to the *Dialogic® NaturalAccess™ OAM API Developer's Manual*.

This topic provides the following information:

- Using the RTP MIB
- Using structured values
- Controlling events

### Using the RTP MIB

---

The Fusion RTP EMC caches data for in-bound and out-bound RTP packet flows. It performs this task for all flows on all CG boards under OAM API management in a chassis. Each MSPP channel is equivalent to either an in-bound flow (RTP IN), out-bound flow (RTP OUT), or a combination of the two (RTP FDX). Each RTP flow is associated with a unique identifier. The syntax is described in this topic.

The Fusion RTP EMC generates OAM events. Each event contains a list of flow identifiers. Clients must use these flow identifiers together with the keyword structure to create a valid keyword name. That keyword name can then be used with the OAM service function **oamGetKeyword** to retrieve information about the flow.

Clients typically access data from the Fusion RTP EMC through two keywords:

RTP EMC keyword	Description
rtp. <b>in-flow-identifier</b> .struct	Accesses inbound RTP simplex channel information.
rtp. <b>out-flow-identifier</b> .struct	Accesses outbound RTP simplex channel information.

The application replaces the bracketed expressions (**in-flow-identifier** or **out-flow-identifier**) with valid flow identifiers to obtain valid values from **oamGetKeyword**. The .struct elements in the keyword names indicate that the value is structured. Refer to *Using structured values* on page 130 for more information.

There are other keywords that do not involve flow identifiers. The names can be taken directly from the table shown RTP EMC keyword hierarchy. Each field separator in a flow identifier is unique, which allows the client to find a particular field quickly with a single call. The flow identifier syntax is as follows:

```

flow-identifier-list:
    flow-identifier
    flow-identifier, flow-identifier-list

flow-identifier:
    in-or-out - dest-udp-port / start-time @ dest-ip-address

in-or-out : one of
            in      out
dest-ip-address :
            n : n : n : n
n :
            0 . . 255
dest-udp-port :
            0 . . 65535
start-time :
            0 . . MAX_UNSIGNED_LONG

```

An ***in-flow-identifier*** begins with in, and an ***out-flow-identifier*** begins with out.

### Using structured values

The Fusion RTP EMC provides keyword values that can be simple or structured. Leaf-node keywords ending with struct have structured values. All other keywords have simple values. Simple values are scalar; structured values are not, and require further explanation.

Structured values are C-style aggregate initializers, without the braces. A structured value is a sequence of one or more fields, separated by commas. They contain no white space. Fields contain simple values, without commas. The syntax for structured values follows:

#### ***structured-value:***

simple-value

simple-value<sub>opt</sub>, structured-value<sub>opt</sub>

#### ***simple-value:***

Any string of characters without commas or white space

The optional values (indicated by opt) in this syntax indicate that there can be any number of empty fields, or empty simple values, in a structured keyword value. Structured values represent the entire tree of values under a keyword formed by removing the .struct element from its name, for example:

**Keyword:** rtp.<***in-flow-identifier***>.local.struct

**Value:** ipAddress,udpPort,SSRC,CNAME,TOOL

Where <***in-flow-identifier***> is replaced by a valid flow identifier and the value is a legal value, usually an integer or a string.

## Controlling events

The Fusion RTP EMC produces two primary events, one to indicate new flows that have just started (RTPEVN\_FLOW\_START), and another to indicate existing flows that have just stopped (RTPEVN\_FLOW\_STOP). These are referred to as flow start and flow stop events. These events are relatively independent of the EMC data cache.

Keyword values for an in-flow or out-flow are valid almost immediately after the flow start event is generated. There is a very small chance that the cache will not have been updated when the client accesses the keyword. In that case, a later attempt will probably obtain a valid value.

Attempts to read keywords for stopped flows (that is, flows for which flow stop events have been returned) always fail.

The relative order between flow start and stop events cannot be preserved, due to the aggregation of flow identifiers into homogeneous lists. In particular, an identifier from a flow start event can have a start time later than an identifier from a following flow stop event.

## Using OAM API functions with the RTP EMC

Applications can use the following functions to work with an OAM API managed object for the RTP MIB extended management component:

OAM API function	Description
<b>oamOpenObject</b>	Opens a managed object and starts an editing session on the specified managed object.  This function returns a handle to be used with subsequent OAM service function invocations. The managed object is completely idle until a client first opens it. Until then, absolutely no caching occurs, and no events are generated. With the first open, caching begins. Subsequent opens do not affect caching in any way.
<b>oamCloseObject</b>	Closes a managed object.  Each close of the managed object, except the last, essentially decrements a reference count. When the last close occurs, the managed object goes completely idle. All caching and events stop, and the cache is erased.
<b>oamGetKeyword</b>	Retrieves a keyword value in the managed object with the specified handle.
<b>oamGetQualifier</b>	Returns information about a specified keyword.

The Fusion RTP EMC is read-only. Therefore, the OAM service function **oamSetKeyword** does not apply and will fail if used.

## RTP EMC object hierarchy

The following table shows RTP EMC fully-qualified object names:

Level 0	Level 0	Level 0	Level 0
Version			
	Major		
	Minor		
pollPeriod			
rtp			
	<b><i>in-flow-identifier</i></b>		
		struct	
		startTime	
		interfaceIndex	
		local	
			struct
			ipAddress
			udpPort
			SSRC
			CNAME
			TOOL
		remote	
			struct
			ipAddress
			udpPort
			SSRC
			CNAME
			TOOL
		jitter	
		numRRSent	
		numSRRcvd	
		timeLastRRSent	
		timeLastSRRcvd	
		lastPTRcvd	
		lostPackets	
		rxPackets	
		rxOctets	
		numBYESRcvd	

Level 0	Level 0	Level 0	Level 0
	<b>out-flow-identifier</b>		
		struct	
		startTime	
		interfaceIndex	
		local	
			struct
			ipAddress
			udpPort
			SSRC
			CNAME
			TOOL
		remote	
			struct
			ipAddress
			udpPort
			SSRC
			CNAME
			TOOL
		jitter	
		numSRSent	
		numRRRCvd	
		timeLastSRSent	
		timeLastRRRCvd	
		lastPTSent	
		RTT	
		lostPackets	
		txPackets	
		txOctets	
		numBYESRcvd	

## Object summary

The following table summarizes the objects that applications can use to retrieve information from the RTP MIB via the RTP EMC (all keywords are read-only):

Task	Use these keywords...
Retrieve version information for the objects	Version Version.Major Version.Minor
Retrieve the polling interval for the RTP EMC	pollperiod 5 [seconds]
Retrieve information about inbound RTP sessions, including the session start time and the associated CG board Ethernet interface	rtp.in-flow-identifier rtp.in-flow-identifier.struct rtp.in-flow-identifier.startTime rtp.in-flow-identifier.interfaceIndex
Retrieve information about the local side of inbound RTP sessions	rtp.in-flow-identifier.local rtp.in-flow-identifier.local.struct rtp.in-flow-identifier.local.ipAddress rtp.in-flow-identifier.local.udpPort rtp.in-flow-identifier.local.SSRC rtp.in-flow-identifier.local.CNAME rtp.in-flow-identifier.remote.TOOL
Retrieve information about the remote side of inbound RTP sessions	rtp.in-flow-identifier.remote rtp.in-flow-identifier.remote.struct rtp.in-flow-identifier.remote.ipAddress rtp.in-flow-identifier.remote.udpPort rtp.in-flow-identifier.remote.SSRC rtp.in-flow-identifier.remote.CNAME rtp.in-flow-identifier.remote.TOOL rtp.in-flow-identifier.jitter[RTP timestamps] rtp.in-flow-identifier.numRRSent rtp.in-flow-identifier.timeLastRRSent rtp.in-flow-identifier.timeLastSRRcvd rtp.in-flow-identifier.lastPTRcvd rtp.in-flow-identifier.rxPackets rtp.in-flow-identifier.rxOctets
Retrieve information about outbound RTP sessions, including the session start time and the associated CG board Ethernet interface	rtp.out-flow-identifier rtp.out-flow-identifier.struct rtp.out-flow-identifier.startTime rtp.out-flow-identifier.interfaceIndex

Task	Use these keywords...
Retrieve information about the local side of outbound RTP sessions	<code>rtp.out-flow-identifier.local</code> <code>rtp.out-flow-identifier.local.struct</code> <code>rtp.out-flow-identifier.local.ipAddress</code> <code>rtp.out-flow-identifier.local.udpPort</code> <code>rtp.out-flow-identifier.local.SSRC</code> <code>rtp.out-flow-identifier.local.CNAME</code> <code>rtp.out-flow-identifier.local.TOOL</code>
Retrieve information about the local side of outbound RTP sessions	<code>rtp.out-flow-identifier.remote</code> <code>rtp.out-flow-identifier.remote.struct</code> <code>rtp.out-flow-identifier.remote.ipAddress</code> <code>rtp.out-flow-identifier.remote.udpPort</code> <code>rtp.out-flow-identifier.remote.SSRC</code> <code>rtp.out-flow-identifier.remote.CNAME</code> <code>rtp.out-flow-identifier.remote.TOOL</code> <code>rtp.out-flow-identifier.numSRSent</code> <code>rtp.out-flow-identifier.timeLastSRSent</code> <code>rtp.out-flow-identifier.timeLastRRRcvd</code> <code>rtp.out-flow-identifier.lastPTSent</code> <code>rtp.out-flow-identifier.RTT[milliseconds]</code> <code>rtp.out-flow-identifier.txPackets</code> <code>rtp.out-flow-identifier.txOctets</code>

## RTP EMC keyword qualifiers

This topic alphabetically lists the RTP EMC keyword qualifiers. Each keyword is first followed by an optional default value and bracket-enclosed units specification. All of the keywords beginning with **rtp** are dynamic, since the user must provide the flow identifier part of them.

Individual objects are described for each keyword qualifier.

### RTP EMC keyword qualifiers

Parameter	Access	Syntax	Description
pollperiod	Read-only	Integer	<p>Polling interval for each board-thread.</p> <p>This keyword indicates the interval at which RTP EMC board-threads poll their board for statistics. On average, the data cached by this RTP EMC cannot update more frequently than this interval.</p> <p>Decimal value between 1 and 31.</p>

### RTP EMC keyword qualifiers

Parameter	Access	Syntax	Description
rtp	N/A	Struct	<p>Top-level keyword. Base of all other dynamic keywords.</p> <p>The flow identifiers in qualifier keywords can be used to retrieve flow statistics, through access to either the keyword <b>rtp.in-flow-identifier</b> or the keyword <b>rtp.out-flow-identifier</b>. The bracket expression must be replaced by a valid flow identifier.</p> <p>Refer to the RTP EMC for a description of the flow identifier syntax. Normally the client obtains flow identifiers from OAM events instead of <b>rtp</b> keyword qualifiers.</p>



---

# 17

## Demonstration programs

---

### MSPP exerciser: **msppxsr**

---

After initializing the system hardware with *oamsys*, run the MSPP exerciser (*msppxsr*) program to verify that MSPP endpoints and channels can be created, connected, and enabled on CG board DSPs. The MSPP exerciser program is a looping demonstration program that repeatedly creates, connects, and enables MSPP channels, then disables, disconnects, and destroys them. It supports the maximum number of connections the host system can support. It also demonstrates sending commands to MSPP endpoint and channel filters.

#### Usage

**msppxsr** *arguments*

#### Featured functions

**mspCreateEndpoint, mspCreateChannel, mspConnect, mspEnableChannel, mspSendCommand, mspDisconnect, mspDisableChannel, mspDestroyChannel**

#### Running **massxsr**

##### Before you begin

Before running *msppxsr*:

- Edit one of the provided CG board keyword files to indicate any system-specific configuration information.
- (Optional) Compile the file *msppxsr.exe* with the make files provided.
- Run *oamsys* with a customized version of one of the sample OAM board keyword files.

## Procedure

To run *msppxsr*:

1. Navigate to the `\nms\fusion\samples\general\msppxsr` directory.
2. Enter the following command:

```
msppxsr -b boardnum ep1 eptype ep2 eptype -c chantype -l localip -d remoteip -nport numport -nloop numloop
```

The following table shows available options. All option expressions must include a space between the option and the value (as shown in the previous example).

Option	Description	Valid entries
-b <b>boardnum</b>	Board number.	Default is 0.
-ep1 <b>eptype</b>	Type of endpoint for endpoint 1. By default, <i>msppxsr</i> creates RTP IPv4 endpoints of the requested type unless you specify the -v6 argument after the <b>eptype</b> argument.	Valid endpoint types include: <ul style="list-style-type: none"> <li>• ds0</li> <li>• rtp (full duplex)</li> <li>• rtpin (simplex receive)</li> <li>• rtpout (simplex receive)</li> <li>• rtpv6 (full duplex)</li> <li>• rtpinv6 (simplex receive)</li> <li>• rtpoutv6 (simplex receive)</li> <li>• tpkt</li> <li>• t38udp</li> <li>• pktmedia</li> </ul>
-ep2 <b>eptype</b>	Type of endpoint for endpoint 2. By default, <i>msppxsr</i> creates RTP IPv4 endpoints of the requested type unless you specify the -v6 argument after the <b>eptype</b> argument.	Default is ds0. See valid entries for ep1 <b>eptype</b> .
-6	Specifies to create RTP IPv6 endpoints of the requested type (full duplex, simplex receive, or simplex send).	
-c <b>chantype</b>	Type of MSPP channel to connect endpoint 1 and 2.	Entry channel type. See <i>Valid channel types</i> on page 139.
-l <b>localip</b>	Local IP address for an RTP or UDP endpoint. If no RTP or UDP endpoint is created, this value is ignored.  The IP address must be appropriate for the type of RTP endpoint specified (that is, an IPv4 address for an IPv4 endpoint or an IPv6 address for an IPv6 endpoint). Entering 0 indicates that the data will not be transferred to an IP network. Use for testing purposes only.	IPv4 or IPv6 address or 0 (for NULL).

Option	Description	Valid entries
-d <b>remoteip</b>	Remote IP address for an RTP or UDP endpoint. If no RTP or UDP endpoint is created, this value is ignored.  The IP address must be appropriate for the type of RTP endpoint specified (that is, an IPv4 address for an IPv4 endpoint or an IPv6 address for an IPv6 endpoint). Entering 0 indicates that the data will not be transferred to an IP network. Use for testing purposes only.	IPv4 or IPv6 address or 0 (for NULL).
-nport <b>numport</b>	Maximum number of channels <i>msppxsr</i> creates.	1 - 360
-nloop <b>numloop</b>	Number of times <i>msppxsr</i> creates and destroys connections. Must be greater than or equal to one.	1 or greater
-nbcast <b>bcastEPs</b>	Number of broadcast (simplex RTP send) endpoints to create plus 1. For example, to create seven broadcast endpoints, specify a <b>bcastEPs</b> of 6.	0 - 7
q	Stops the program.	NA
?	Displays on screen Help.	NA

For example:

```
msppxsr -b 1 -ep1 rtp -ep2 ds0 -c 4 -l 0 -d 0 -nport 1 -nloop 10
```

The previous example commands *msppxsr* to create set up and tear down one duplex G.729A/B voice connection (made up of a DS0 endpoint, a G.729A/B channel, and an RTP IPv4 endpoint) ten times. No data is transferred to the IP network because the local IPv4 address and remote IPv4 address are each set to 0.

*msppxsr* displays results of MSPP function calls at the command prompt and displays status information about MSPP connections.

Before *msppxsr* sets up and tears down MSPP connections for the final time (iteration **numloop** - 1), it displays a numbered list of commands you can invoke for the connected channels. The list varies according to the kinds of channels created.

3. Enter the number associated with any commands to execute.

*msppxsr* executes the specified command.

4. Stop the program after entering commands by entering **Q**.

### Valid channel types

Value	Description
1	G.711 full duplex
2	G.723 full duplex
3	G.726 full duplex
4	G.729 full duplex
5	Fax relay full duplex
6	G.711 encode simplex
7	G.711 decode simplex
8	G.723 encode simplex

Value	Description
9	G.723 decode simplex
10	G.726 encode simplex
11	G.726 decode simplex
12	G.729 encode simplex
13	G.729 decode simplex
14	G.728 full duplex
15	G.728 encode simplex
16	G.728 decode simplex
17	RTP switch simplex
19	AMR full duplex
20	AMR encode simplex
21	AMR decode simplex
44	EVRC full duplex
45	EVRC encode simplex
46	EVRC decode simplex
47	ILBC 2.0 full duplex
48	ILBC 2.0 encode simplex
49	ILBC 2.0 decode simplex
50	ILBC 3.0 full duplex
51	ILBC 3.0 encode simplex
52	ILBC 3.0 decode simplex
53	GSM-FR full duplex
54	GSM-FR encode simplex
55	GSM-FR decode simplex

## Description

*msppxsr* creates and enables the number of MSPP channels specified (**numport**) for the number of iterations specified (by **numloop**). The command line interface shows the progress of MSPP commands as they execute. *msppxsr* uses **ctaCreateQueue**, **ctaCreateContext**, and **ctaOpenService** to create CTA queues, contexts, open CTA services for MSPP endpoints and channels. *msppxsr* then creates, connects, and enables MSPP endpoints and channels with **mspCreateEndpoint**, **mspCreateChannel**, **mspConnect**, and **mspEnableChannel**.

After the **numport** channels have been enabled, *msppxsr* disables, disconnects, and destroys the MSPP channels with **mspDisableChannel**, **mspDisconnect**, **mspDestroyChannel**, and **mspDestroyEndpoint**. *msppxsr* exercises each MSPP channel the number of times specified by **numloop**.

When *msppxsr* stops running in the middle of a cycle, an expected event has not been received because of a configuration error or software mismatch. Check configuration files, runtime files, and component file dates and make sure the PATH environment variable does not redirect the system to an older file.

## Fusion nailed-up sample: *msppsamp*

---

*msppsamp* demonstrates MSPP universal ports by setting up as many as 360 channels for voice or fax data transfer and can perform voice to fax switchover. Each MSPP channel processes and encodes/decodes voice and fax data. It transfers the data between PSTN (DS0) and IP (RTP or UDP) endpoints.

Running *msppsamp* performs the following tasks:

- Creates the maximum supported number of MSPP connections between endpoints at the IP network and the PSTN interfaces.
- Responds to incoming calls from the PSTN and enables MSPP connections.
- Provides a mechanism for controlling and querying the filters that make up MSPP connections.
- Initializes and loads the CG board (using *oamsys*) when invoked through the sample command file *demo.cmd*.

For an overview of an *msppsamp* test environment, refer to the *msppsamp illustration* on page 29.

This topic contains the following information:

- *msppsamp* overview
- Running *msppsamp*
- Details of operation

### *msppsamp* overview

---

Sets up and enables MSPP connections to transfer data from voice and fax calls across the gateway from one network to another.

#### Usage

```
demo
```

#### Featured functions

***mspCreateEndpoint*, *mspCreateChannel*, *mspConnect*, *mspEnableChannel*, *mspSendCommand*, *mspDisconnect*, *mspDisableChannel*.**

*msppsamp* also uses NaturalAccess functions to initialize the MSPP API, and to set up event queues and contexts. It uses the NCC API to perform call control at the PSTN interface.

## Requirements

*msppsamp* requires the test equipment to place and receive voice (and optionally, fax) calls over both CG board interfaces.

## Running msppsamp

### Before you begin

Before running the demonstration program:

1. Edit the *msppsamp demo.cmd* file to specify the following information:

Entry	Description	Allowed values
-b <b>boardnum</b>	CG board number. Default is 0.	N/A
-6	Specifies that the connection will be conducted over an IPv6 link (without this argument, the default is IPv4).	N/A
-d <b>destinationIPaddress</b>	IPv4 or IPv6 address of remote gateway where the voice or fax data is transferred.	IPv4 or IPv6 address.
-D <b>DTMFinband</b>	Enable or disable inband DTMF carriage. Default is 0.	0 - Disable inband DTMF carriage 1 - Enable inband DTMF carriage
-E	Disables software echo cancellation processes performed on board DSPs. Set this option when using hardware echo cancellation (CG 6x6x Series boards only).	N/A
-f <b>numfaxcalls</b>	Total number of fax channels for the program to run.	CG 6565C = Up to 480 ports CG 6565/E = Up to 240 ports CG 6060/C = Up to 120 ports
-g <b>vocodertype</b>	Type of encoder/decoder filter to use. For fax, the -g field is ignored. Use commas with the supported vocoder types in the following combinations: <ul style="list-style-type: none"> <li>• Entering a single <b>vocodertype</b> without commas specifies a full duplex path that uses the specified encoder and decoder.</li> <li>• Inserting a comma before the <b>vocodertype</b> (,711) specifies a simplex channel that uses only the voice decoder.</li> <li>• Inserting a comma after the <b>vocodertype</b> (723,) specifies a simplex channel that uses only the voice encoder.</li> <li>• Separating different <b>vocodertypes</b> with a comma (723, 711) specifies a duplex channel that uses one type of encoder (G.723) with another decoder (G.711).</li> </ul>	711 723 726 729 728 AMR EVR GFR L20 L30

Entry	Description	Allowed values
-i 0xxxxx	Specifies what events to display. Default is 0x0013.	bit 0 - All errors. bit 1 - NCC events for call setup and tear down. bit 2 - NCC miscellaneous events. bit 4 - All MSPP commands and events. bit 5 - MSPP Fax billing events. bit 6 - MSPP and RTCP Report events.
-l <b>localIPaddress</b>	Local IPv4 or IPv6 address of the CG board Ethernet interface.	IPv4 or IPv6 address.
-p <b>protocol</b>	Call control protocol to use (for example nocc, isdn, mfc2, or wnk0).	Any protocol supported under Natural Access.
-r <b>RTCPsetting</b>	Enable or disable RTCP report receiving and event generation. Default is 0.	0 = Disable RTCP report events. 1 = Enable RTCP report events.
-t <b>startingtimeslot</b>	Starting timeslot and port number. Default is 0.	0 to 119 for E1 boards. 0 to 95 for T1 boards.
-L <b>IUDPportnum</b>	Local UDP port to use as a base for the first voice channel. Unless another address is specified, <b>IUDPportnum</b> defaults to the address specified for the remote UDP port (the <b>rUDPportnum</b> specified with the -u option). The next channel increments this UDP port number by two. Therefore, for the default port, the first channel uses port 2000, and the next one uses 2002, then 2004, and so on.	UDP port number.
-u <b>rUDPportnum</b>	Remote UDP port to use as a base for the first voice channel. The next channel increments this UDP port number by two. Therefore, for the default port, the first channel uses port 2000, and the next one uses 2002, then 2004, and so on.	UDP port number.
-U <b>tUDPportnum</b>	UDP port to use as a base for the first T.38 fax channel. The next channel increments this UDP port number by two. Therefore, for the default port, the first voice channel uses port 2000, and the next one uses 2002, then 2004, and so on.	UDP port number.

Entry	Description	Allowed values
-n <b>bcastEPs</b>	Number of broadcast (simplex RTP send) endpoints to create plus 1. For example, to create seven broadcast endpoints, specify a <b>bcastEPs</b> of 6.	0 to 7
-v <b>numvoicecalls</b>	Total number of voice channels for the program to run. Default is 1. The total number of voice and fax calls cannot exceed 120.	1 to 120.
-m <b>modemtransport</b>	Enables modem or fax (DCE transport over a clear channel. Default is 0.	0: Disables DCE transport 1: Enables DCE transport
-g <b>framequota</b>	Number of frames per packet. For more information about frame quotas, refer to the <i>Dialogic® NaturalAccess™ Fusion™ VoIP API Developer's Manual</i> .	Default is 2.
-j <b>jitterdepth</b>	Jitter buffer depth. For more information about jitter depth, refer to the <i>Dialogic® NaturalAccess™ Fusion™ VoIP API Developer's Manual</i> .	Default is 2 x <b>framequota</b> .

2. Edit one of the sample Fusion CG board keyword files so that it contains configuration information appropriate for the system.
3. Edit the *fusion\_oamsys.cfg* file so that it specifies the following information for the system:

Entry	Description
Bus	Bus number of the CG board.
Slot	Slot number of the CG board.
File	File name of the CG board keyword file.

Running the *msppsamp* command file (*demo.cmd*) automatically boots the CG board (using *oamsys*) according to the information specified in the modified *fusion\_oamsys.cfg* file and the modified sample Fusion CG board keyword file.

## Procedure

To run *msppsamp*:

1. Navigate to the *nms\fusion\samples\msppsamp* directory.
2. Edit the *demo.cmd* file (as described in Before you begin) to specify appropriate parameters for your system configuration and for desired program operation.
3. Enter the following command:

```
demo
```

4. Use test equipment to establish phone calls to the Fusion gateway system line interfaces. All incoming calls from the CG board are routed to the specified destination IP address.

*msppsamp* displays information about its activities as it performs data transfer and conversion tasks.



5. Enter any of the following options at the prompt. These commands take effect for all active ports:

Option	Sub-option	Description
A	Voice encoder filter commands.	
	1	Increase voice encoder filter gain.
	2	Decrease voice encoder filter gain.
	3	Enable or disable Voice Activity Detection (VAD).
	4	Set IP side formatting for voice encoder filter (G.711 only).
	5	Enable or disable DTMF/CED suppression filters.
	6	Set encoding mode to online or offline.
	7	Set encoder filter rate.
	8	Change RTP payload ID in encoder filter.
	9	Set DTMF mode.
	S	Query voice encoder filter state.
	P	Print voice encoder filter state.
B	RTP endpoint filter commands.	
	1	Enable or disable RTCP events.
	2	Enable or disable link status events.
	3	Set number of frames per RTP packet.
	4	Change RTP payload ID in RTP endpoint and voice decoder filters.
	5	Change expected payload ID for DTMF packets.
	6	Control MSPP DTMF events.
	S	Query RTP endpoint state.
	P	Print RTP endpoint filter state.
C	Jitter filter commands.	
	1	Increase jitter buffer depth.
	2	Decrease jitter buffer depth.
	3	Enable or disable adaptive jitter mode.
	S	Query jitter filter state.
	P	Print jitter filter state.

Option	Sub-option	Description
D	Voice decoder filter commands.	
	1	Increase voice decoder filter gain.
	2	Decrease voice decoder filter gain.
	3	Set IP side formatting for voice decoder filter (G.711 only).
	4	Set decoding mode online or offline.
	5	Change RTP payload ID on decoder and RTP endpoint.
	6	Set DTMF mode.
	7	Play DTMF digit.
	S	Query voice decoder filter state.
	P	Print voice decoder filter state.
E	Echo commands.	
	1	Enable or disable echo cancel adaptation (performed through the ADI API).
	2	Reset echo taps enable or disable (performed through the ADI API).
	3	Enable or disable echo suppressor (performed through the ADI API).
	4	Enable or disable echo cancel bypass (performed through the ADI API).
	5	Query echo canceller status.
F	CG board monitoring commands.	
	1	Query board Ethernet status.
	2	Query board CPU usage.
	3	Query the list of all IPv6 addresses.
1	T.38 fax relay filter commands.	
	1	T.38 fax relay filter configuration.
	2	T.38 fax diagnostics.
	S	T.38 fax relay filter query
	P	Print fax relay session status.
2	T38UDP endpoint filter commands.	
	1	T.38 IFP encode configuration.
	2	T.38 UDPTL encode configuration.
	3	T.38 IFP decode configuration.
	4	T.38 UDPTL decode configuration.
	S	Query T38UDP endpoint state.
	P	Print T38UDP endpoint state.

Option	Sub-option	Description
S		Show or hide DPF event data (internal use only).
	1	Suppress event data.
	2	Show event data.

6. Press the **ESC** key to exit the program.

### Details of operation

*msppsamp* is a nailed up application because it performs no IP-side call control. Call data from each PSTN port is automatically directed to an IP address specified in the CG board keyword file used with *oamsys*.

*msppsamp* uses a single thread to wait for PSTN Call Control (NCC) events and asynchronous MSPP events (like the CED\_DETECT event used in voice/fax switchover).

In addition, *msppsamp* creates a single NaturalAccess event queue to receive events for the following objects:

Objects	Description
MSPP API objects (endpoints and channels)	<i>msppsamp</i> creates three contexts per dedicated voice or fax channels (one channel, two endpoints), or five contexts per voice/fax switchover channel (two channels, three endpoints).
NCC API objects (lines and calls)	<i>msppsamp</i> creates one context per port for call control.

When you run *demo.cmd*, the program:

1. Initializes NaturalAccess.
2. Creates a NaturalAccess event queue.
3. Creates all MSPP endpoints and channels.
4. Enables MSPP channels as calls come in (MSPP endpoints are enabled by default).
5. Starts the CED tone detector that is needed for voice to fax switchover and transfers voice data across the channel.

For voice/fax switchover, the application waits for a fax CED tone from the PSTN endpoint. When a CED is detected, *msppsamp* sends a command to the DS0 endpoint to switch from voice to fax data transfer, disables the voice channel, and enables the fax channel.

For information about configuring T.38 fax filters, refer to the *Dialogic® NaturalAccess™ Media Stream Protocol Processing API Developer's Manual*.

Switching from fax data back to voice data is not supported. Consequently, when a voice channel is switched to fax with the -v option in the *demo.cmd* file, the channel must remain a fax channel.

## RTP switch sample: rtpswitchsamp

---

The RTP Switch sample program (*rtpswitchsamp*) creates a switching or forking bridge that consists of two full duplex connections switched together and an arbitrary number of monitoring parties attached to each end as described in Details. Switching from RTP IPv4 endpoints to RTP IPv6 endpoint (and vice versa) is not supported.

This sample program demonstrates RTP switching capability in Fusion.

### Usage

```
sw.cmd
```

### Featured functions

**mspCreateEndpoint, mspCreateChannel, mspConnect, mspEnableChannel, mspDisableChannel, mspDisconnect**

### Before you begin

Before running the sample program:

1. Edit one of the sample Fusion CG board keyword files so that it contains configuration information appropriate for the system.
2. Edit the *fusion\_oamsys.cfg* file so that it specifies the following information for the system:

Entry	Description
Bus	Bus number of the CG board.
Slot	Slot number of the CG board.
File name	File name of the CG board keyword file.

3. Edit the *sw.cmd* file to specify the following information:

Entry	Description	Allowed values
-l <b>localIPAddress</b>	Local IPv4 or IPv6 address of the CG board Ethernet interface (for example, IP address of Gateway 1 in the following illustration).	IPv4 or IPv6 address.
-d <b>destIPAddress1</b>	Destination IPv4 or IPv6 address of the gateway where the voice data will be transmitted (for example, IP address of Gateway 2 in the following illustration).	IPv4 or IPv6 address.
-E	Disables echo cancellation processes performed on board DSPs. Set this option when using hardware echo cancellation on CG 6565 and CG 6565C boards.	N/A
-r <b>destIPAddress2</b>	Destination IPv4 or IPv6 address of the gateway where the voice data will be transmitted (for example, IP address of Gateway 3 in the following illustration).	IPv4 or IPv6 address.
-6	Specifies to create RTP IPv6 endpoints (the default is RTP IPv4 endpoints).	
-u <b>udpportnum</b>	UDP port to use as a base for the first channel. Default is 5004.	UDP port number.
-m <b>udpportmon</b>	UDP port to use as a base for the first monitoring channel. Default is 5040).	UDP port number.
-n <b>numfork</b>	Number of forking bridges to be created. Default is 1.	1 - 11.
-b <b>boardnum</b>	CG board number. Default is 0.	Not applicable.

**Example**

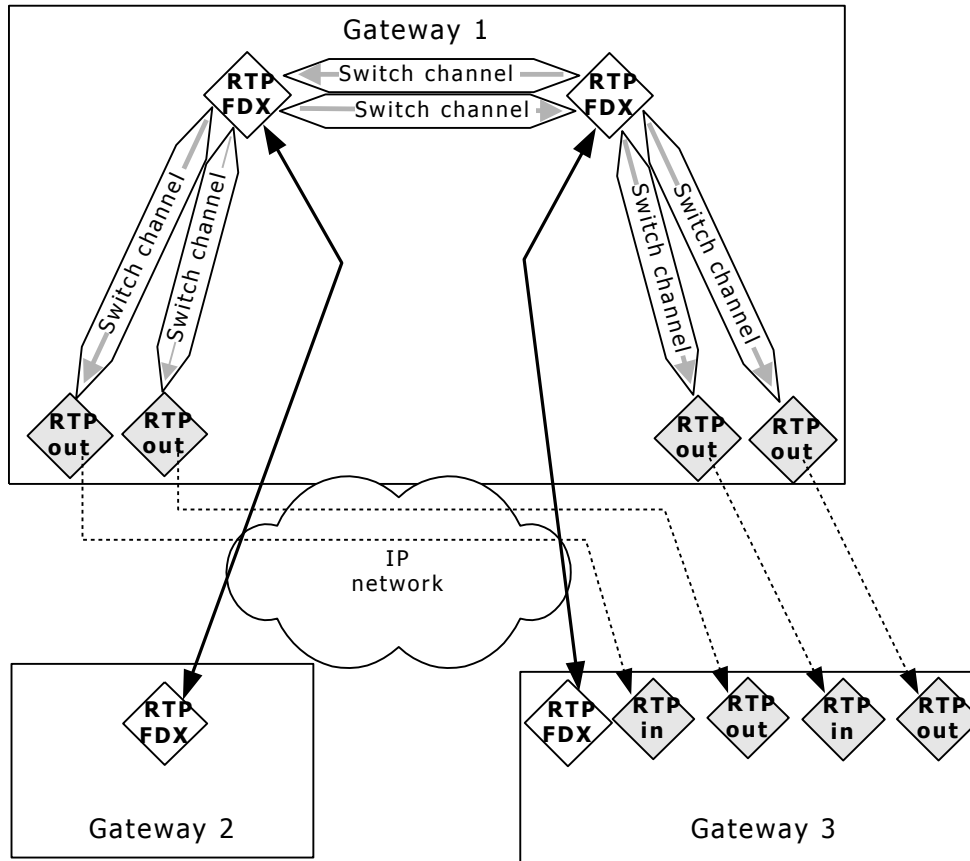
The following command creates two forking bridges on the board. *rtpswitchsamp* creates, connects, and enables the RTP IPv4 endpoints required to create the bridges. It then waits for the user to press a key and then disables and destroys the bridge.

```
Rtpswitchsamp -l 198.62.150.1 -d 198.62.150.2 -r 198.62.150.3 -u 5004 -m 5040 -n 2 -b 0
```

## Details

After initializing the system hardware with *oamsys*, run *rtpswitchsamp* to verify that the system supports switching between RTP connections.

As shown in the following illustration, the RTP Switch sample program creates a switching or forking bridge that consists of two full duplex connections switched together, and fifteen monitoring parties attached to each end:



In this configuration, RTP out endpoints on Gateway 1 act as monitoring endpoints that direct intercepted voice data to Gateway 3.

## Native play and record test: natprdemo

*natprdemo* demonstrates:

- Initializing NaturalAccess and creating multiple contexts.
- Creating an MSPP channel with RTP and DS0 endpoints.
- Using ADI API and Voice message API functions to play and record RTP media streams without transcoding, and collecting RFC2833 digits on the MSPP channel.
- Passing RFC 2833 digits to the ADI digit collection function.

### Usage

```
natprdemo -R remoteIPAddress -r remoteportnum options
```

where

- ***remoteIPAddress*** is the IPv4 address of the remote device,
- ***remoteportnum*** is the UDP port of the remote device,
- ***options*** include the following:

Option	Description
-b <b><i>boardnum</i></b>	Board number. Default is 0.
-e <b><i>recordencoder</i></b>	Type of encoding for record. Default is 46 (EDTX mu-law). Valid values include: 46 = EDTX mu-law 47 = EDTX A-law 48 = EDTX G.726 ADPCM 49 = EDTX G.729a 50 = EDTX G.723 153 = EDTX AMR
-p <b><i>playencoder</i></b>	Type of encoding for play. Default is 10. Valid values include: 10 = mu-law11 = A-law 20 = 32 bit G.726 ADPCM 46 = EDTX mu-law 47 = EDTX A-law 48 = EDTX G.726 ADPCM 49 = EDTX G.729a 50 = EDTX G.723 153 = EDTX AMR
-f <b><i>filename</i></b>	Name of the file to which you want to direct recorded data. Default is <i>temp.vox</i> .
-L <b><i>localIPAddress</i></b>	IPv4 address of the local computer. Default is 127.0.0.1.
-l <b><i>localportnum</i></b>	UDP port to use on the local computer. Default is the same port number as the remote port.
-q	Quiet mode. This option suppresses most console messages.

Option	Description
-s <i>timeslot</i>	IVR timeslot number. Default is 0.
-d <i>value</i>	<p>Decode Record stream flag. Valid values include:</p> <ul style="list-style-type: none"> <li>• 1 = Use decoder</li> <li>• 0 = Do not use decoder</li> </ul> <p>You must enable the decoder (the default setting) if you are implementing silence detection.</p>

## Description

*natprdemo* uses the MSPP API and NaturalAccess to implement an interactive voice response (IVR) application for IP telephone calls. It establishes a two-way voice path from the local computer to a remote IP address and port. The demonstration program does not set up the call or obtain the remote address. In an actual application, call setup is handled by a separate protocol such as the session initiation protocol (SIP).

This program demonstrates synchronous programming on a single port. After each command, *natprdemo* retrieves events continuously until the expected event returns.

## Procedure

Run *natprdemo* by using a telephone connected to a full duplex G.711 RTP stream at a known IP address and port (for example, a gateway that uses a CG board).

To run *natprdemo*:

Step	Action
1	Navigate to the <code>\nms\fusion\samples\natprdemo</code> directory in Windows or the <code>/opt/nms/fusion/samples/natprdemo</code> directory in Linux.
2	Enter the following command: <code>natprdemo -R <i>remoteIPAddress</i> -r <i>remoteportnum</i></code>
3	From the telephone, listen for the greeting. After playing the greeting, <i>natprdemo</i> displays a menu of options.
4	From the telephone, press 1.
5	Record a brief voice message.
6	Press 2.
7	Verify that the recorded message plays back.
8	Close <i>natprdemo</i> by pressing 3.



## Native play and record test: natprtest

---

The Native play record test (*natprtest*) program verifies that the system supports creating and enabling MSPP connections and ADI ports and demonstrate NMS native play and record functionality. This sample program demonstrates NaturalAccess and Fusion functions executing in asynchronous mode.

### Featured functions

NaturalAccess, Voice Message API, MSPP API, and ADI API functions.

### Purpose

Use *natprtest* to:

- Verify proper installation and operation of the NCC API and the ADI service.
- Experiment with the NaturalAccess native play record feature.
- Demonstrate working examples of NaturalAccess and NCC API or ADI API functions.
- Demonstrate creating, connecting, enabling, disabling, disconnecting, and destroying MSPP endpoints and channels.
- Demonstrate sending commands to MSPP endpoint and channel filters.

## Usage

*natprtest* [**options**]

where valid options include:

Option	Description
A <b>xxxmgr</b>	The NaturalAccess service manager. Default: ADIMGR
-b <b>n</b>	The board number <b>n</b> . Default: 0
-C <b>filename</b>	Load a keyword configuration from a text file. Keywords supported are: <ul style="list-style-type: none"> <li>• Board=<b>n</b> Where <b>n</b> specifies the board number.</li> <li>• Stream=<b>n</b> Where <b>n</b> specifies the logical stream.</li> <li>• Slot=<b>n</b> Where <b>n</b> specifies the logical timeslot.</li> <li>• Msp.localIP=<b>xx.xx.xx.xx</b> Specifies the local IP address for MSPP endpoints.</li> <li>• Msp.remoteIP=<b>xx.xx.xx.xx</b> Specifies the remote IP address for MSPP endpoints.</li> <li>• Msp.LocalUdpPort=<b>port</b> Where <b>port</b> specifies the local UDP port for MSPP endpoints.</li> <li>• Msp.UdpPort=<b>port</b> Where <b>port</b> specifies the remote UDP port for MSPP endpoints.</li> <li>• msp_channelType=<b>chantype</b> Where <b>chantype</b> is enumerated as defined below.</li> <li>• PayloadId=<b>pid</b> Where <b>pid</b> specifies the RTP payload ID to use for MSPP endpoint egress packets.</li> </ul>
-f <b>filename</b>	The file name to retrieve service names. Default: None
-F <b>filename</b>	The name of the NaturalAccess configuration file. Default: <i>cta.cfg</i> . The configuration file must specify to initialize the msp service and msgmgr service manager.
-i <b>filename</b>	The name of the input file. This option allows you to use a file that lists NaturalAccess test commands and data line by line, instead of entering the commands interactively.
-l	Show low-level events.
-p <b>protocol</b>	Protocol to run. Default is nocc.
-s [ <b>stream:</b> ] <b>slot</b>	The port (DSP) address. Default is 0:0.
-e <b>localip</b>	Local IP address for an RTP or UDP endpoint. If no RTP or UDP endpoint is created, this value is ignored. The IP address must be appropriate for the type of RTP endpoint specified (that is, an IPv4 address for an IPv4 endpoint or an IPv6 address for an IPv6 endpoint). Entering 0 indicates that the data will not be transferred to an IP network. Use for testing purposes only.
-r <b>remoteip</b>	Remote IP address for an RTP or UDP endpoint. If no RTP or UDP endpoint is created, this value is ignored. The IP address must be appropriate for the type of RTP endpoint specified (that is, an IPv4 address for an IPv4 endpoint or an IPv6 address for an IPv6 endpoint). Entering 0 indicates that the data will not be transferred to an IP network. Use for testing purposes only.

Option	Description
-c <b>chantype</b>	Type of MSPP channel to connect endpoints. Valid channel types include: 1 G.711 full duplex 2 G.723 full duplex 3 G.726 full duplex 4 G.729 full duplex 5 Fax relay full duplex 6 G.711 encoder simplex 7 G.711 decoder simplex 8 G.723 encoder simplex 9 G.723 decoder simplex 10 G.726 encoder simplex 11 G.726 decoder simplex 12 G.729 encoder simplex 13 G.729 decoder simplex 14 G.728 full duplex 15 G.728 decoder simplex 16 G.728 encoder simplex 17 RTP switching simplex
-t <b>tracemode</b>	Enables or disables tracing. d = Disable tracing r = Enable tracing (default)
-v	Set the NaturalAccess compatibility level to 0.
-w	Wait before exit.
-?	Access help.

## Description

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

The following table describes the available commands:

Function	Command	Description
Help	H	Displays a table of available commands.
Quit	Q	Exits from <i>natprtest</i> .
Repeat command	!	Repeats the previous command.
Abort dial	AD	Calls <b>adiStopDial</b> .
Abort timer	AT	Calls <b>adiStopTimer</b> .
Accept call	CC	Calls <b>nccAcceptCall</b> .

Function	Command	Description
Answer call	AC	Calls <b>nccAnswerCall</b> .
Assert signal	AS	Calls <b>adiAssertSignal</b> .
Attach context	AX	Calls <b>ctaAttachContext</b> .
Attach object	AO	Calls <b>ctaAttachObject</b> .
Block calls	BC	Calls <b>nccBlockCalls</b> .
Call progress begin	CB	Calls <b>adiStartCallProgress</b> .
Call progress stop	CS	Calls <b>adiStopCallProgress</b> .
Call status	C?	Calls <b>nccGetCallStatus</b> .
Change context	CH	Switches between different contexts created by <i>natprtest</i> with the CX or PX commands.
Close services (new)	CV	Calls <b>ctaCloseServices</b> .
Close services (old)	CP	Calls <b>ctaDestroyContext</b> .
Collect digits	CD	Calls <b>adiCollectDigits</b> .
Collect stop	SC	Calls <b>adiStopCollection</b> .
Get context information	CN	Calls <b>ctaGetContextInfoEx</b> .
Continuous event fetch	CE	Enables fetching of events.
Create context	CX	Calls <b>ctaCreateContextEx</b> .
Create persistent context	PX	Calls <b>ctaCreateContextEx</b> with the CTA_CONTEXT_PERSISTENT flag set.
Create void context	VX	Calls <b>ctaCreateContext</b> with ctaqueuehd set to NULL_CTAQUEUEHD.
Destroy context	DX	Calls <b>ctaDestroyContext</b> .
Digit flush	DF	Calls <b>adiFlushDigitQueue</b> .
Digit get	DG	Calls <b>adiGetDigit</b> .
Digit peek	DP	Calls <b>adiPeekDigit</b> .
Disable bit detector	DB	Calls <b>adiStopSignalDetector</b> .
Disable DTMF detector	DD	Calls <b>adiStopDTMFDetector</b> .
Disable energy detector	DE	Calls <b>adiStopEnergyDetector</b> .
Disable MF detector	DM	Calls <b>adiStopMFDetector</b> .
Disable tone detector	DT	Calls <b>adiStopToneDetector</b> .
Disconnect call	DC	Calls <b>nccDisconnectCall</b> .
Detach object	DO	Calls <b>ctaDetachObject</b> .
Enable bit detector	EB	Calls <b>adiStartSignalDetector</b> .
Encoding information	EI	Calls <b>vceGetEncodingInfo</b> .
Set native play and record information	SN	Calls <b>adiSetNativeInfo</b> .
Enable DTMF detector	ED	Calls <b>adiStartDTMF</b> .
Enable energy detector	EE	Calls <b>adiStartEnergyDetector</b> .

Function	Command	Description
Enable MF detector	EM	Calls <b>adiStartMFDetector</b> .
Enable tone detector	ET	Calls <b>adiStartToneDetector</b> .
Enumerate contexts	EC	Lists contexts created by <i>natprtest</i> with the CX, PX, or VX commands.
Find file	FF	Calls <b>ctaFindFile</b> .
Format event	FE	Calls <b>ctaFormatEventEx</b> .
FSK abort receive	FA	Calls <b>adiStopReceivingFSK</b> .
FSK receive	FR	Calls <b>adiStartReceivingFSK</b> .
FSK send	FS	Calls <b>adiStartSendingFSK</b> .
Generate DTMFs	GD	Calls <b>adiStartDTMF</b> .
Generate net tone	GN	Calls <b>adiStartTones</b> .
Generate wink	GW	Calls <b>adiStartPulse</b> .
Generation stop	GS	Calls <b>adiStopTones</b> .
Generate user tone	GT	Calls <b>adiStartTones</b> .
Get call status	L?	Calls <b>nccGetLineStatus</b> .
Get context list	CL	Calls <b>ctaQueryServerContexts</b> .
Get event source	GM	Calls <b>ctaGetEventSources</b> .
Get object descriptor	DH	Calls <b>ctaGetObjDescriptor</b> .
Get one event	GE	Enables fetching of one event.
Get parameter ID	PI	Calls <b>ctaGetParmID</b> .
Get server version	RV	Calls <b>ctaGetVersionEx</b> .
Get service version	SV	Calls <b>ctaGetServiceVersionEx</b> .
Hold call	OC	Calls <b>nccHoldCall</b> .
Loopback event	LE	Calls <b>ctaQueueEvent</b> .
Modify play gain	MG	Calls <b>vceSetPlayGain</b> .
Modify play speed	MS	Calls <b>vceSetPlaySpeed</b> .
Open services (new)	OV	Calls <b>ctaOpenServices</b> .
Open services (old)	OP	Calls <b>ctaCreateContext</b> and <b>ctaOpenServices</b> .
Place call	PC	Calls <b>nccPlaceCall</b> .
Play file	PF	Calls <b>vceOpenFile</b> and <b>vcePlayMessage</b> .
Play memory	PM	Calls <b>vcePlayMessage</b> .
Play/receive status	P?	Call <b>vceGetContextInfo</b> .
Play/receive stop	PS	Calls <b>vceStop</b> .
Play/record status	R?	Calls <b>vceGetContextInfo</b> .
Play/record stop	RS	Calls <b>vceStop</b> .
Query capability	CQ	Calls <b>nccQueryCapability</b> .

Function	Command	Description
Query services	ES	Calls <b>ctaQueryServices</b> .
Record file	RF	Calls <b>vceCreateFile</b> and <b>vceRecordMessage</b> .
Record memory	RM	Calls <b>vceCreateMemory</b> , <b>vceEraseMessage</b> , and <b>vceRecordMessage</b> .
Refresh parameters	RP	Calls <b>ctaRefreshParms</b> .
Reject call	JC	Calls <b>nccRejectCall</b> .
Release call	RC	Calls <b>nccReleaseCall</b> .
Retrieve call	RR	Calls <b>nccRetrieveCall</b> .
Send digits	ND	Calls <b>nccSendDigits</b> .
Server shutdown	SH	Calls <b>ctaShutdown</b> .
Set call handle	IH	Specifies active and held call handles.
Set default server	SS	Calls <b>ctaSetDefaultServer</b> .
Set event handler	EH	Calls <b>ctaSetErrorHandler</b> .
Set event source	SM	Calls <b>ctaSetEventSources</b> .
Set parameters	PD	Calls <b>ctaGetParmByName</b> and <b>ctaGetParmInfo</b> .
Set server trace	TR	Calls <b>ctaSetGlobalTraceMask</b> .
Set timeslot	TS	Specifies a timeslot to open the NCC service.
Show state	X	Calls <b>adiGetContextInfo</b> and <b>nccGetLineStatus</b> .
Signal bit query	SQ	Calls <b>adiQuerySignalState</b> .
Start dial	SD	Calls <b>adiStartDial</b> .
Start protocol	SP	Calls <b>nccStartProtocol</b> or <b>adiStartProtocol</b> .
Start timer	ST	Calls <b>adiStartTimer</b> .
Stop event fetch	SE	Disables continuous fetch of events.
Stop protocol	UP	Calls <b>nccStopProtocol</b> or <b>adiStopProtocol</b> .
Supervised transfer	VT	Calls <b>nccTransferCall</b> .
Toggle pattern	TP	Calls <b>adiAssertSignal</b> .
Trace	W	Calls <b>ctaSetTraceLevel</b> .
Unblock calls	UC	Calls <b>nccUnblockCalls</b> .
View parameters	VD	Calls <b>ctaGetParmID</b> and <b>ctaGetParmInfoEx</b> .
Open MSPP port	MO	Calls <b>ctaCreateContext</b> and <b>ctaOpenServices</b> for the MSPP service.
Close MSPP port	MC	Calls <b>ctaDestroyContext</b> .
Create RTP Endpoint	MR	Calls <b>mvpCreateEndpoint</b> with <b>MSP_ENDPOINT_RTPFDX</b> endpoint type.
Create DS0 endpoint	MX	Calls <b>mvpCreateEndpoint</b> with <b>MSP_ENDPOINT_DS0</b> endpoint type.
Create MSPP channel	MH	Calls <b>mvpCreateChannel</b> .
Connect MSPP channel	MN	Calls <b>mvpConnect</b> .

Function	Command	Description
Enable MSPP channel	MA	Calls <b>mspEnableChannel</b> .
Disable MSPP channel	MB	Calls <b>mspDisableChannel</b> .
Get MSPP filter handle	MF	Calls <b>mspGetFilterHandle</b> .
Set marker string	MM	Prompts for a string prefix for the next command response in the log file.

You can enter commands while asynchronous functions execute, allowing you to execute multiple asynchronous functions concurrently or to stop functions. For example, you can run a tone detector (ET) and record voice (RF) simultaneously. Abort any of these functions by entering the respective stop command (DT and RS for tone and record). You can view NaturalAccess parameters (VD) or modify NaturalAccess parameters (PD). If *ctdaemon* is running, shared system global defaults are copied to the context. Otherwise, the local compiled defaults are copied. The commands prompt you to view or modify context parameters or global defaults.

*natprtest* writes trace messages to *ctdaemon* if *ctdaemon* is running. Use *ctdaemon* or **ctaSetTraceLevel** (W command) to set the global or local trace mask. To loop back an application event, use **ctaQueueEvent** (LE command).

### Programming notes

*natprtest* includes functions for retrieving commands from the keyboard and executing them. These functions include:

Function	Description
<b>PerformFunc</b>	Executes the keyboard commands and initiates Natural Access, NCC service, or ADI API functions.
<b>Main</b>	Processes options, creates NaturalAccess event queue, and goes into <b>ctaWaitEvent</b> loop.
<b>MyPlayAccess</b>	Voice play callback.
<b>MyRecAccess</b>	Voice record callback.
<b>MyEventHandler</b>	Displays the event and performs post-event processing (for example, closes files, prints extra information).

## ThroughPacket sample: tpktsamp

*tpktsamp* demonstrates creating, connecting, enabling ThroughPacket connections for transferring data across the gateway. *tpktsamp* requires the test equipment to place and receive voice calls over CG board interfaces.

### Featured functions

**mspCreateEndpoint, mspCreateChannel, mspConnect, mspEnableChannel, mspSendCommand, mspDisconnect, mspDisableChannel.**

*tpktsamp* also uses Natural Access functions to initialize the MSPP service, and to set up event queues and contexts and uses the NCC service to perform call control at the PSTN interface.

### Usage

demo

## Running tpktsamp

### Before you begin

Before running the demonstration program:

1. Edit the *tpktsamp* configuration file (*demo.cmd*) to specify the following information:

Entry	Description	Allowed values and default
-b <b>boardnum</b>	CG board number. Default is 0.	N/A
-p <b>ccprotocol</b>	Call control protocol to use.	Default is wnk0.
-c <b>tpktmode</b>	ThroughPacket session delivery mode to use.	complex = Payloads from different sessions are combined into larger packets.  simple = Separate media sessions are carried in separate packets.
-E	Disables echo cancellation processes performed on board DSPs. Set this option when using hardware echo cancellation on CG 6565 and CG 6565C boards.	N/A
-l <b>localIPaddress</b>	Local IP address of the CG board Ethernet interface.	IPv4 or IPv6 address. There is no default local IP address.
-l <b>localgtwaddress</b>	Remote L1 gateway address.	IPv4 or IPv6 address. here is no default remote L1 gateway address
-u <b>UDPportnum</b>	Voice media (RTP) UDP port number. The next channel increments this UDP port number by two. Therefore, for the default port, the first channel uses port 50000, and the next one uses 50002, then 50004, and so on.	UDP port number.



Entry	Description	Allowed values and default
-t <b>numcalls</b>	Number of ThroughPacket voice calls to start.	Default is 1.
-v <b>numcalls</b>	Number of RTP voice calls to start.	Default is 0.
-s <b>timeslot</b>	Timeslot at which to start.	Default is 0.
g <b>vocodertype</b>	Type of encoder/decoder filter to use. Use commas with the supported vocoder types in the following combinations: <ul style="list-style-type: none"> <li>• Entering a single <b>vocodertype</b> without commas specifies a full duplex path that uses the specified encoder and decoder.</li> <li>• Inserting a comma before the <b>vocodertype</b> (,711) specifies a simplex channel that uses only the voice decoder.</li> <li>• Inserting a comma after the <b>vocodertype</b> (723,) specifies a simplex channel that uses only the voice encoder.</li> <li>• Separating different <b>vocodertypes</b> with a comma (723, 711) specifies a duplex channel that uses one type of encoder (G.723) with another decoder (G.711).</li> </ul>	G723 G711 G726 G729
-e <b>flag</b>	A flag to indicate whether to enable RTCP. Use only with RTCP sessions.	0 = Disabled (default). 1 = Enabled.

2. Edit one of the sample Fusion CG board keyword files so that it contains configuration information appropriate for the system.
3. Edit the *fusion\_oamsys.cfg* file so that it specifies the following information for the system:

Entry	Description
Bus	Bus number of the CG board.
Slot	Slot number of the CG board.
File	File name of the CG board keyword file.

Running the *tpktsamp* command file (*demo.cmd*) automatically boots the CG board according to the information specified in the modified *fusion\_oamsys.cfg* file and the modified sample Fusion CG board keyword file.

## Procedure

To run *tpktsamp*:

Step	Action																				
1	Navigate to the <i>nms\fusion\samples\tpktsamp</i> directory.																				
2	Edit the <i>demo.cmd</i> file (as described in Before you begin) to specify appropriate parameters for your system configuration and for desired program operation.																				
3	Enter the following command: <div>demo</div>																				
4	Use test equipment to establish phone calls to the Fusion gateway system line interfaces. All incoming calls from the CG board are routed to the specified destination IP address. <i>tpktsamp</i> displays information about the its activities as it performs data transfer and conversion tasks.																				
5	Enter any of the following options at the prompt. These commands take effect for all active ports: <table border="1"> <thead> <tr> <th>Option</th><th>Description</th></tr> </thead> <tbody> <tr> <td>A</td><td>Increase jitter buffer depth by 1.</td></tr> <tr> <td>B</td><td>Decrease jitter buffer depth by 1.</td></tr> <tr> <td>C</td><td>Query jitter state.</td></tr> <tr> <td>D</td><td>Query encoder state.</td></tr> <tr> <td>E</td><td>Query decoder state.</td></tr> <tr> <td>F</td><td>Query ThroughPacket and RTP state.</td></tr> <tr> <td>G</td><td>Disable ThroughPacket filters.</td></tr> <tr> <td>H</td><td>Reconfigure and re-enable ThroughPacket filters.</td></tr> <tr> <td>1</td><td>Map payload ID 127 for G723Low vocoder.</td></tr> </tbody> </table> <p>Encoder and decoder filters can receive queries only when they are enabled.</p>	Option	Description	A	Increase jitter buffer depth by 1.	B	Decrease jitter buffer depth by 1.	C	Query jitter state.	D	Query encoder state.	E	Query decoder state.	F	Query ThroughPacket and RTP state.	G	Disable ThroughPacket filters.	H	Reconfigure and re-enable ThroughPacket filters.	1	Map payload ID 127 for G723Low vocoder.
Option	Description																				
A	Increase jitter buffer depth by 1.																				
B	Decrease jitter buffer depth by 1.																				
C	Query jitter state.																				
D	Query encoder state.																				
E	Query decoder state.																				
F	Query ThroughPacket and RTP state.																				
G	Disable ThroughPacket filters.																				
H	Reconfigure and re-enable ThroughPacket filters.																				
1	Map payload ID 127 for G723Low vocoder.																				
6	Press the <b>ESC</b> key to exit the program.																				

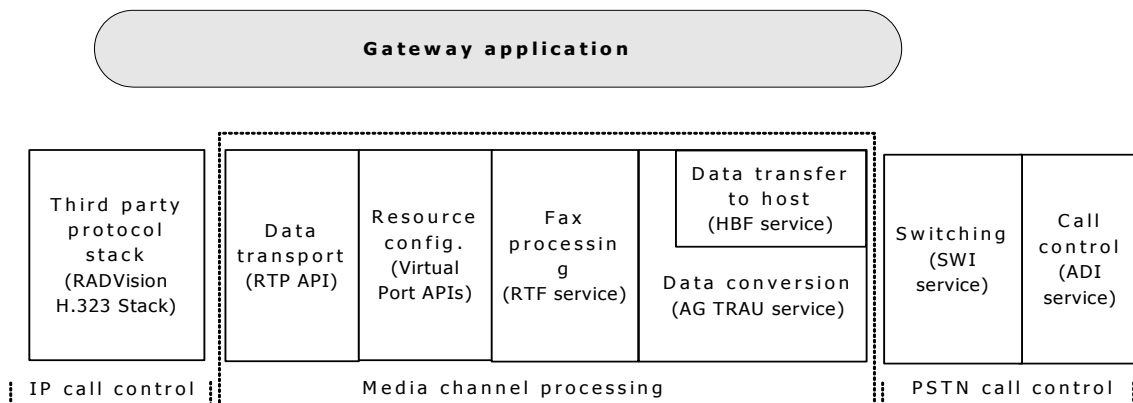
# 18 Fusion migration

## Migrating from Fusion 3 to Fusion 4

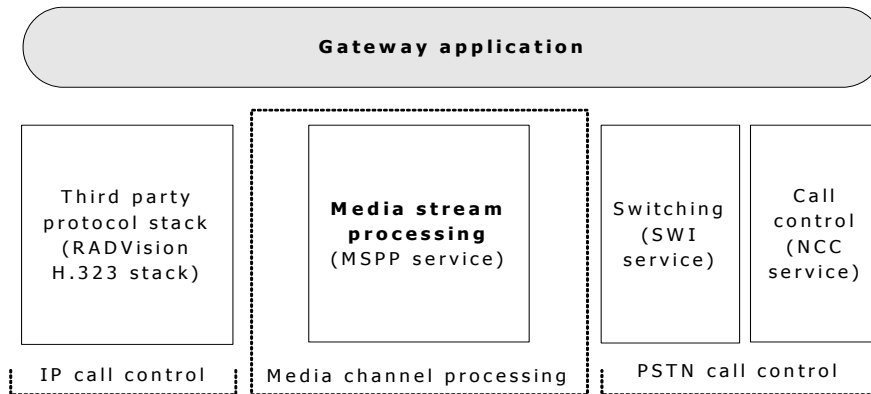
Fusion 4 provides extended functionality and simplified programming models from earlier versions of Fusion. Fusion 4 extended features include:

Feature	Fusion 4	Earlier Fusion releases
End-to-end gateway media channel construction and switching	Performed transparently by the MSPP service	Switching performed explicitly by application through the SWI service and the TX APIs.
Creation of connection endpoints for PSTN, host, and IP network interfaces	Performed through MSPP service	Performed through AG TRAU, HBF service, and Virtual Port APIs.
Runtime control of media channel processing (for example, fax, G.711, or G.723.1.)	Performed through MSPP service	Performed through AG TRAU, HBF service, and Virtual Port APIs.
Interfaces for both the IP network and PSTN	CG board provides both network interfaces	AG line interface board provides PSTN interface. TX board provides IP network interface
Integrated PSTN and IP call control	Performed by using a third party IP protocol stack in conjunction with Natural Access PSTN call control	Performed by using a third party IP protocol stack in conjunction with Natural Access PSTN call control

The following illustration shows a broad overview of the earlier Fusion 3 programming model:



The following illustration shows the simplified Fusion 4 programming model. Notice that all of the media processing tasks, formerly performed with several different APIs and services, are now accomplished with the MSPP service only:



In both models, application initialization, PSTN call control, and switching are controlled by Natural Access and its services (ADI, NCC, and SWI services). Call control at the IP network interface can be performed through a third party protocol stack such as the SIP or H.323.

The following table shows the operations required to create and control an end-to-end media channel from a PSTN or an IP network in a Fusion 4 and Fusion 3 application:

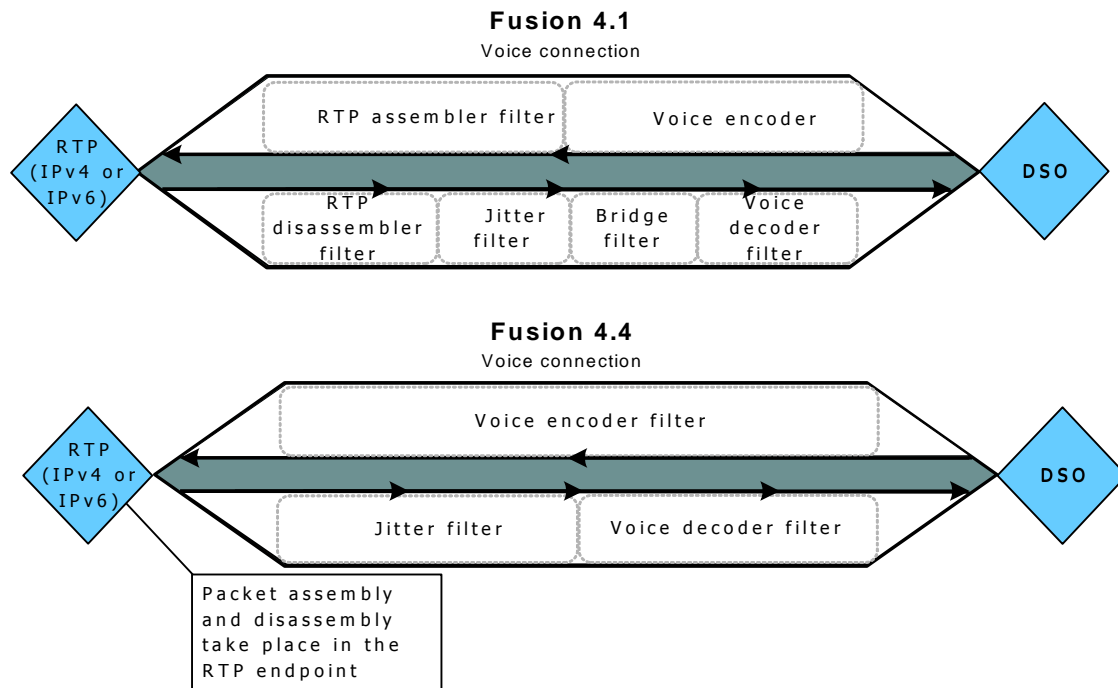
Gateway application media stream tasks	Fusion 4.4 application	Fusion 3.0 application
Respond to incoming PSTN calls	Natural Call Control (NCC) service	ADI service
Switch PCM data from AG or CG Series board line interfaces to AG or CG Series board ports	Set up during board configuration or performed manually (with the SWI service)	SWI service
Switch voice/fax data from PSTN ports to board DSPs	Performed transparently by the MSPP service	SWI service
Control media channel data processing	MSPP service	AG TRAU service
Transfer data to host	MSPP service	HBF service
Transfer data from board resource back to CT bus	Performed transparently by the MSPP service	SWI service
Transfer data from CT bus to IP interface	Performed transparently at board level	TPX API, Common Management API, and CPI library
Set up and control media channel endpoints at IP interface	MSPP service	Virtual Port API and RTP API
Perform IP call control	Third party protocol stack	Third party protocol stack

## Migrating to Fusion 4.4

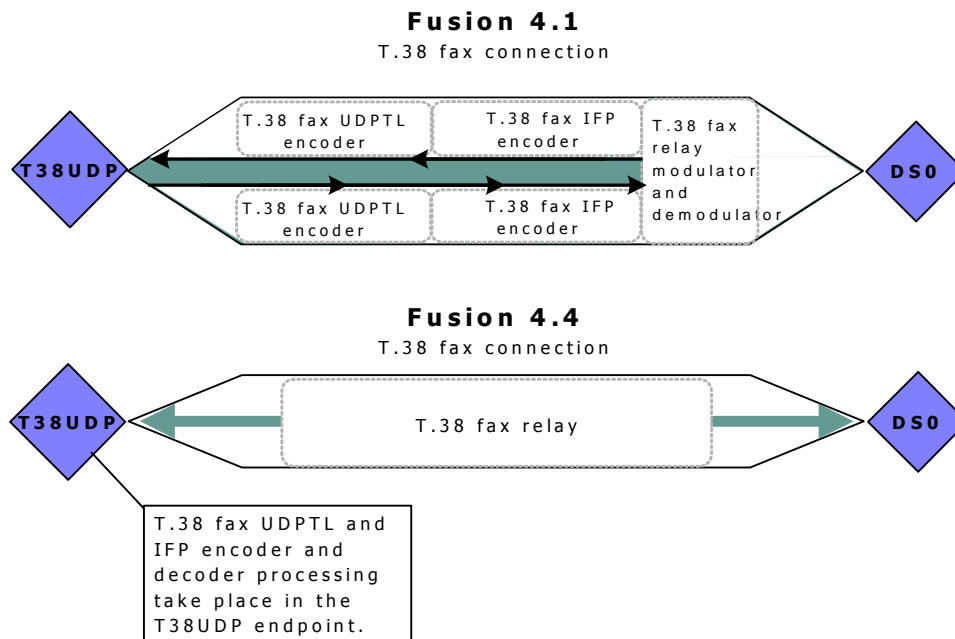
Fusion 4.4 provides substantial improvements and new features over Fusion 4.1. New features simplify the programming model and enhance the processing efficiency of MSPP connections. The changes include new and modified MSPP endpoint and channel types and new structures for creating and configuring these MSPP service objects. These changes require developers to make minor code modifications and to recompile existing applications.

### Channel enhancements

To improve CG board CPU resource usage, Fusion 4.4 reduces the number of filters to transfer data through MSPP connections. This is accomplished by moving some voice and fax processing tasks from MSPP channels to MSPP endpoints. For example, Fusion 4.1 voice channels performed voice data packetization and de-packetization through assembler and dissembler filters. In Fusion 4.4, voice data packetization and de-packetization takes place within the RTP endpoint. The following illustration shows the filter chains used by voice channels in Fusion 4.1 and the filter chains used by voice channels in Fusion 4.4:



In Fusion 4.4, simplex voice connections, either the voice decoder or voice encoder filter is omitted from the voice channel (depending on the direction of the data transfer). The following illustration shows the difference between T.38 fax connections in Fusion 4.1 and Fusion 4.4:



The following table shows the filters that make up MSPP service channels in Fusion 4.1 and Fusion 4.4:

Channel	Fusion 4.1 channel filters	Fusion 4.4 channel filters
Duplex voice	Bridge, jitter, RTP assembler, RTP disassembler, voice decoder, voice encoder	Jitter, voice encoder, voice decoder
T.38 fax	IFP decoder, IFP encoder, T.38 fax modulator and demodulator, UDPTL decoder, UDPTL decoder	T.38 fax relay

### Configurable defaults

In Fusion 4.1, when an application created an endpoint, it specified parameters that customize the way the endpoint processed data. However, an application could only configure MSPP service channel parameters after creating a channel by sending configuration commands to the individual filters.

In Fusion 4.4, all channels and endpoints provide parameter structures so applications can specify the processing characteristics of the MSPP service object when creating the object. Therefore the application can specify initialization parameters for any data processing task, whether the processing is performed by an endpoint or channel.

The following table shows the channel parameter structures added for Fusion 4.4, as well as the parameters (formerly applicable to MSPP channels in Fusion 4.1), that were moved to endpoint structures.

MSPP channel	Parameter structure	Configurable items	Parameters moved to endpoint structure
Voice channel	MSP_VOICE_CHANNEL_PARMS	<ul style="list-style-type: none"> <li>Jitter processing</li> <li>Voice encoder and decoder</li> </ul>	Packet assembly and disassembly
T.38 fax relay	MSP_FAX_CHANNEL_PARMS	T.38 fax modulation and demodulation	<ul style="list-style-type: none"> <li>IFP encoding and decoding</li> <li>UDPTL encoding and decoding</li> </ul>

### Other Fusion 4.4 Features

In addition to channel enhancements and configurable defaults, Fusion 4.4 provides a wide range of new features for developing VoIP gateway and media server applications. These features include:

Feature	Description	For information, see...
Support for IPv6 through RTP IPv6 endpoints	Enables CG boards to implement IPv6 capabilities (for example, address autoconfiguration, neighbor discovery, and IP security) supported in Natural Access 2003-1.	<i>Dialogic® NaturalAccess™ Media Stream Protocol Processing API Developer's Manual</i> and CG board manual.
Inband DTMF carriage	Implements the RFC 2833 specification by enabling MSPP voice channels to detect DTMF digits in voice data received from PSTNs and transfer DTMF signaling information within RTP packets. The payload contained in each packet contains only information about the type of tone detected, the tone's gain, and the tone's duration.	<i>Transferring DTMF digits according to RFC 2833</i> on page 75.
RTP switching channels	Enables applications to perform legal intercept and call monitoring tasks by switching voice data between RTP IPv4 and IPv6 endpoints.	<i>RTP forking and RTP switching</i> on page 55.
RTP loopback	Enables applications to reverse the direction in which voice data flows when the data reaches a RTP endpoint. The data traverses the CG board IPv4 or IPv6 stack and is looped back at the IP layer so that it can be returned to a DS0 endpoint.	<i>Loopback connections</i> on page 61.
ThroughPacket endpoints	Provides a mechanism for reducing Fusion gateway bandwidth by multiplexing voice data payloads from multiple voice channels into larger packets.  IPv6 is not supported for ThroughPacket connections.	<i>ThroughPacket multiplexing overview</i> on page 115.
Adaptive jitter	Sets the jitter filter depth automatically and dynamically according to the frequency with which the jitter filter is receiving frames. When the application enables adaptive jitter (it is disabled by default), the jitter buffer increases or decreases the number of frames in the jitter depth according to the number of frames received in the previous five seconds.	<i>Dialogic® NaturalAccess™ Media Stream Protocol Processing API Developer's Manual</i> .





# Index

## A

adiStartCallProgress 52

ANS 101

## B

before using Fusion 25

board keywords 26, 31, 31, 32, 36, 120

boards (CG) 15

booting 27

configuration 26, 31, 31, 36, 39

queries 71, 72, 72

utilities 28

## C

call control 49, 50, 52

calling card services 56

CED tone 105, 107, 109, 112

cg6kcon 28

cg6kfusion.cfg 137

cgroute 28, 36

channels 21

clear channels (DCE over IP) 102

creating 44

destroying 45

inband DTMF carriage 80

switch 60, 61, 61, 62

T.38 fax 109

CNG tone 112

commands 45

channel 82

endpoint 127

compiling and linking 30

configurable defaults 165

connections 20, 21

and call control 49

loopback 61, 61, 62

setting up 44

ThroughPacket 126

voice/fax 103, 107

contexts 11

CPU utilization queries 72

ctaCreateContext 43, 77

ctaCreateQueue 43

ctaInitialize 43

ctaOpenServices 43

ctaSetParmByName 52

## D

data rate 117

DCE over IP transport 97, 100, 101, 102

demonstration programs 29, 137, 141, 148

DLM 120

DPFs 31, 77

DPM 31

DS0 loopback connections 61, 61

DTMF 13, 15, 69

## E

echo cancellation 101

echo\_v4.f54 97

endpoints 23

creating 44

customizing RTP endpoint behavior 83

destroying 45

ThroughPacket (TPKT) 116, 123, 126

Ethernet interfaces 15, 36, 65, 72

## F

f54info 31

fax 97, 103

- DCE 100, 101, 102
- T.38 105, 107, 109, 113
- fax billing events 113
- filters 23, 45
- frames 81, 102, 126, 141, 167
- Fusion 13
  - demonstration programs 137, 141, 148
  - getting started 25
  - installation 25, 28
  - migration 163, 165
  - requirements 15
- fusion\_oamsys.cfg 26, 27, 31

**G**

- G.711 15, 97, 102
- G.723.1 15, 75, 126
- G.726 15
- G.729A 15

**H**

- H.248 105
- H.323 15, 163

**I**

- IFP packets 103
- inband DTMF 69, 79, 82
- installation 25
- IP addresses 36
- ITU T.38 103

**J**

- jitter 67, 102, 104, 118, 141, 165

**K**

- keywords 26, 31
  - configuring DSP resources 31
  - CT bus clocking 26
  - DCE over IP 100
  - DSP 31
  - echo canceller 100
  - IP interface 26, 36
  - line interface 26

- mediamask 35, 77
- resource 33, 77, 100
- sample configuration files 26
- ThroughPacket 120

**L**

- linking 30
- loopback connections 61, 61, 62

**M**

- media server 13
- mediamask 35, 77
- MEGACO 105
- MGCP 105
- migration 163, 165
- Monitoring applications 58
- mspBuildCommand 127
- mspBuildQuery 71, 128
- mspcmd.h 30
- mspConnect 44, 50, 52, 107, 126
- mspCreateChannel 44, 79
- mspCreateEndpoint 44, 123
- mspdef.h 30
- mspDestroyChannel 45
- mspDestroyEndpoint 45
- mspDisableChannel 45
- mspDisableEndpoint 45, 47
- mspDisconnect 45
- mspEnableChannel 44, 50
- mspEnableEndpoint 44
- mspinit.h 30
- mspobj.h 30
- MSPP API 19
  - and NCC call control 52
  - function sequence 47
  - setting up connections 44
  - tearing down connections 45
- msppsamp 29, 141
- msppxsr 28, 137
- mspquery.h 30

mspReleaseBuffer 45, 69, 113  
mspSendCommand 45, 113, 124, 126, 127  
mspSendQuery 45, 71, 128  
mspunsol.h 30, 113

## N

native play and record 85  
    demonstration programs 151, 153  
    performing native play 86  
    performing native record with decoding 88  
    performing native record without decoding 92  
natprdemo 151  
natprtest 153  
NaturalAccess 15, 17, 25, 43  
NCC API 50, 52  
nccAcceptCall 50  
nccAnswerCall 50  
nccPlaceCall 52  
nccRejectCall 50  
nccStartProtocol 77

## O

OAM configuration 26, 31, 31, 32, 36, 131  
oamsys 27, 29, 31, 36

## P

packet rate 118  
pciscan 27

## Q

queries 45  
    endpoint 128  
    system 71, 72, 72

## R

resource keywords 33, 39, 77, 100  
RFC 2833 75

    configuring compliant connections 77, 79, 80, 82  
    transferring non-DTMF RFC 2833 events 77  
    unsolicited events 69

route availability events 65, 73

RTCP report events 67

RTP EMC MIB 132

RTP forking 55

RTP loopback connections 61, 62

RTP switching 55

    applications 56, 58

    limitations 60

    overview 20

rtpswitchsamp 148

## T

T.30 fax 105

T.38 fax 103

    channels 105

    unsolicited events 113

    voice to fax switchover 107, 109, 112

ThroughPacket 115

    configuring board resources 120

    multiplexing 115, 117, 118, 119

    ThroughPacket sample program 160

    TPKT endpoints 116, 123, 124, 126, 127

tpktsamp 160

## U

UDP ports 36, 65, 123

UDPTL 103

unsolicited events 65, 65, 67, 71, 73

utilities 28

## V

voice/fax switchover 107, 109