Intel® Dialogic® System Release 6.0 CompactPCI* Feature Pack 1 for Windows*

Release Update

December 20, 2007

05-2373-031

Publication Date: December 20, 2007

Document Number: 05-2373-031

Intel
1515 Route 10
Parsippany, NJ 07054

For **Technical Support**, visit the Intel Telecom Support Resources website at:
http://developer.intel.com/design/telecom/support

For **Products and Services Information**, visit the Intel Telecom and Compute Products website at:
http://www.intel.com/design/network/products/telecom

For **Sales Offices** and other contact information, visit the Buy Telecom Products page at:
http://www.intel.com/buy/networking/telecom.htm

About This Publication

This section contains information about the following topics:

Purpose

This Release Update addresses issues associated with Intel® Dialogic® System Release 6.0 CompactPCI* Feature Pack 1 for Windows*. In addition to summarizing issues that were known as of the Release's general availability, it is intended that this Release Update will continue to be updated to serve as the primary mechanism for communicating new issues, if any, that may arise after the release date.

Intended Audience

This Release Update is intended for users of Intel Dialogic System Release 6.0 CompactPCI Feature Pack 1 for Windows.

How to Use This Publication

This Release Update is organized into four sections (click the section name to jump to the corresponding section):

- Document Revision History: This section summarizes the ongoing changes and additions that are made to this Release Update after its original release. This section is organized by document revision and document section.
- Post-Release Developments: This section describes significant changes to the system release subsequent to the general availability release date. For example, the new features provided in Service Updates are described here.
- Release Issues: This section lists issues that may affect the system release hardware and software. The primary list is sorted by issue type, but alternate sorts by defect number, by product or component, and by Service Update number are also provided.
- Documentation Updates: This section contains corrections and other changes that apply to the System Release documentation set that were not made to the documents prior to the release. The updates are organized by documentation category and by individual document.

Related Information

See the following for additional information:

- For information about the products and features supported in this release, see the *Intel Dialogic System Release 6.0 cPCI Feature Pack 1 for Windows Release Guide*, which is included as part of the documentation set for this release.
- For further information on issues that have an associated defect number, you may use the Defect Tracking tool at *http://membersresource.dialogic.com/defects/*. When you select this link, you will be asked to either LOGIN or JOIN.
- *http://www.dialogic.com/support/* (for Dialogic technical support)
- *http://www.dialogic.com/* (for Dialogic® product information)

Document Revision History

This Revision History summarizes the changes made in each published version of the Release Update for Intel® Dialogic® System Release 6.0 CompactPCI* Feature Pack 1 for Windows*, which is a document that has been and is intended to be periodically updated throughout the lifetime of the release.

Document Rev 31 - published December 20, 2007

Updated for Service Update 83.

In the Post-Release Developments section:

- Updated the information about Mixing ISDN and Clear Channel on a DMN160TEC or DMT160TEC Board to indicate that CAS is supported on the DMT160TEC board but not on the DMN160TEC board (IPY00038087).
- Deleted function reference information for **dcb_SetPartyParm( )** and **dcb_GetPartyParm( )** under Enhanced Volume Control for Conferencing, because this information has been incorporated into the updated *Audio Conferencing API Library Reference* that is now on the online documentation bookshelf.

In the Release Issues section:

- Added the following resolved problems: IPY00041280, IPY00041296.
- Eliminated the link to view issues sorted by PTR number. (PTR numbers have been superseded by defect numbers. The PTR numbers still appear in the Release Issues table for historical purposes, but a version of the table sorted by PTR number is no longer provided.)

In the Documentation Updates section:

- Deleted the corrections for the Audio Conferencing API Library Reference and Standard Runtime Library API Programming Guide, because these corrections have been incorporated into updated documents that are now on the online documentation bookshelf. (Other documents with updated versions on the online documentation bookshelf are: Audio Conferencing API Programming Guide, Continuous Speech Processing API Programming Guide, Continuous Speech Processing API Library Reference, and Standard Runtime Library API Library Reference.)
- Added documentation updates to the Voice API Programming Guide and Voice API Library Reference for functions that are no longer supported (**r2_creatfsig( )** and **r2_playbsig( )**).

Document Rev 30 - published August 1, 2007

Updated for Service Update 82.

In the Release Issues section, added the following resolved problems: IPY00037391, IPY00038280, IPY00038533, IPY00038572, IPY00038708, IPY00038849, IPY00038956, IPY00038979, IPY00039163.

In the Documentation Updates section:

- Added updates to the *Intel NetStructure for DM3 Architecture for cPCI on Windows Configuration Guide* about NFAS D channel backup (DCBU) supported on 4ESS, 5ESS, and NI-2.
- Added an update to the *Fax Software Reference* about the default fax font (IPY00037855).

Document Rev 29 - published June 4, 2007

Updated for Service Update 78.

In the Release Issues section, added the following resolved problems: IPY00037302, IPY00038365.

Document Rev 28 - published May 24, 2007

Updated for Service Update 77.

In the Release Issues section, added the following resolved problems: IPY00036855, IPY00037632, IPY00037767, IPY00037918, IPY00038060, IPY00038074.

In the Documentation Updates section, added an update for the **gc_InitXfer( )** function under *Global Call API Library Reference* (IPY00038401).

Document Rev 27 - published May 7, 2007

Updated for Service Update 76.

In the Release Issues section, added the following resolved problems: IPY00037619, IPY00037864, IPY00037989.

Document Rev 26 - published March 28, 2007

Updated for Service Update 75.

In the Post-Release Developments section, deleted some of the detailed descriptions about diagnostics features that were previously included in this section, because this information is now superseded by the updated *Diagnostics Guide* that is now on the documentation bookshelf.

In the Release Issues section, added the following resolved problems: IPY00034429, IPY00034816, IPY00035148, IPY00035451, IPY00036248, IPY00036919, IPY00037356, IPY00037432.

In the Documentation Updates section, deleted the relevant corrections for the *Diagnostics Guide* because these corrections have been incorporated into an updated document that is now on the documentation bookshelf.

Document Rev 25 - published March 15, 2007

Updated for Service Update 73.

In the Post-Release Developments section, added information about binary log files to the Enhanced Runtime Trace Facility (RTF) section (IPY00037518).

In the Release Issues section, added the following resolved problems: IPY00033640, IPY00036025, IPY00037351, IPY00037372.

Document Rev 24 - published February 12, 2007

Updated for Service Update 71.

In the Release Issues section, added the following resolved problems: IPY00033164, IPY00034406, IPY00034559, IPY00034627, IPY00034841, IPY00035831.

Document Rev 23 - published September 20, 2006

Updated for Service Update 68.

In the Release Issues section, added the following resolved problems: IPY00006790 (35137), IPY00031561 (36755), IPY00033698, IPY00034404.

Document Rev 22 - published August 29, 2006

Updated for Service Update 64.

In the Post-Release Developments section:

- Under Global Call Support for Time Slots on SS7 Boards Running in DTI Mode, deleted the restriction that opening trunk devices is not supported. Trunk devices can be opened.
- Under Notification of Layer 1 Alarm Events on SS7 Boards, revised the Alarm Handling for SS7 Boards section to indicate that GCEV_ALARM events are disabled by default and must be enabled via gc_SetAlarmConfiguration( ).

In the Release Issues section, added the following resolved problems: IPY00006816 (36737), IPY00033472, IPY00033763.

Document Rev 21 - published August 7, 2006

Updated for Service Update 62.

In the Post-Release Developments section, under New Features in Global Call Protocols Package, added five more new protocols (Bulgaria R2, Croatia R2, Kuwait R2, Lithuania R2, Uzbekistan R2) and new parameters for Nortel Meridian Lineside E1 protocol.

In the Release Issues section, added the following resolved problem: IPY00006077 (PTR 36237).

In the Documentation Updates section:

- Added an update to the NCM_ApplyTrunkConfiguration( ) function under *Native Configuration Management API Library Reference*.
- Added IPY00006540 (PTR 34211) under *Global Call ISDN Technology Guide*.
- Added IPY00006580 (PTR 34546) and IPY00006537 (PTR 35666) under *Voice API Programming Guide*.

Document Rev 20 - published June 19, 2006

Updated for Service Update 59.

In the Post-Release Developments section, added the following:

- Global Call Support for Time Slots on SS7 Boards Running in DTI Mode.
- Windows Services Control Panel Name Changes.
- Notification of Layer 1 Alarm Events on SS7 Boards.
- Automatic FCD File Generation.
- Centralized Logging using Runtime Trace Facility (RTF).
- New OAMIPC Mechanism Replaces CORBA.

In the Release Issues section, added the following resolved problems: IPY00029931 (PTR 36809), IPY00031597 (PTR 36527), IPY00032271 (PTR 36699), IPY00032954, IPY00033005.

In the Documentation Updates section, made changes to the following:

- In the *Intel Dialogic System Release 6.0 cPCI Feature Pack 1 for Windows Software Installation Guide*, added information about deleting references to CORBA.
- In the *Intel NetStructure for DM3 Architecture for cPCI on Windows Configuration Guide*:
  - Added information about deleting references to fcdgen utility.
  - Added documentation updates about the PhysicalSlotNumber and PciID parameters.
- In the *Intel Dialogic System Release 6.0 on cPCI for Windows Administration Guide*, added information about the DCM menu option Start Server Only Mode.
- In the *Intel Dialogic System Software for DM3 Architecture Products on Windows Diagnostics Guide*, added information to the Runtime Trace Facility (RTF) Reference chapter.

- In the *Global Call API Library Reference*, added information about new GCAMS functions for SS7 technology.
- In the *Global Call SS7 Technology Guide*, added information for alarm handling for SS7 boards and about Global Call support for time slots on SS7 boards running in DTI mode.
- In the DCM Online Help, added documentation updates about the PhysicalSlotNumber and PciID parameters.

Document Rev 19 - published May 31, 2006

Updated for Service Update 57.

In the Release Issues section, added the following resolved problem: IPY00032363.

Document Rev 18 - published May 10, 2006

Updated for Service Update 55.

In the Release Issues section, added the following resolved problem: IPY00032996.

Document Rev 17 - published April 20, 2006

Updated for Service Update 54.

In the Release Issues section:

- Added the following resolved problems: IPY00006712 (PTR 36790), IPY00006846 (PTR 36711), IPY00028547 (PTR 35670), IPY00032262 (PTR 36688), IPY00032265 (PTR 36780). Also added IPY00028649 (PTR 36416) (fixed in Service Update 50).
- Added the following known problem: IPY00032702.

Document Rev 16 - published April 12, 2006

Updated for Service Update 53.

Note: The Release Issues section has been modified to show issues by Change Control System defect number and by PTR number. Issues reported prior to March 27, 2006, will be identified by both numbers. Issues reported after March 27, 2006, will only have a defect number.

In the Post-Release Developments section:

- Added Enhanced Runtime Trace Facility (RTF).
- Added Intel Telecom Subsystem Summary Tool (its_sysinfo) Command Line Execution.
- Added Windows Server 2003 R2 under New Operating System Support.

In the Release Issues section:

- Added the following resolved problem: IPY00011037 (PTR 36677).
- Added the following known problem: IPY00032271 (PTR 36699).
- Added the following known (permanent) problem: IPY00029958 (PTR 36722).

In the Documentation Updates section:

- Added documentation updates to the following documents because of new features in the Service Update: *Intel Dialogic System Release 6.0 cPCI Feature Pack 1 for Windows Release Guide*, *Intel Dialogic System Software for DM3 Architecture Products on Windows Diagnostics Guide*.
- Added IPY00031917 (PTR 27337) under *Fax Software Reference*.
- Added IPY00029956 (PTR 36646) under *Global Call IP Technology Guide*. Also added documentation update about the IP_H221NONSTANDARD data structure.
- Added documentation update to the *Modular Station Interface API Programming Guide* about using HDSI boards.
- Added documentation update to the *Standard Runtime Library API Programming Guide* about performance considerations.

Document Rev 15 - published March 14, 2006

Updated for Service Update 51.

In the Post-Release Developments section:

- Added Call Rejection Notification for Blocked Channel Recovery.
- Added Windows 2000 Update Rollup 1 for SP4 under New Operating System Support.

In the Release Issues section, added the following resolved problems: 33780, 35748, 36186, 36296, 36667. Also added 34180 (fixed in Service Update 47).

In the Documentation Updates section:

- Added PTR# 32157 under *Event Service API Library Reference*.
- Added documentation updates to the following documents because of new features in the Service Update: *Intel Dialogic System Release 6.0 cPCI Feature Pack 1 for Windows Release Guide*, *Global Call ISDN Technology Guide*

Document Rev 14 - published February 28, 2006

Updated for Service Update 50.

In the Post-Release Developments section, revised the installation information under Service Update for System Release 6.0 CompactPCI Feature Pack 1 to indicate that the Modify and Change options can now be used.

In the Release Issues section, added the following resolved problems: 32316, 36298, 36413, 36483, 36584, 36633, 36647. Also added 36196 (fixed in Service Update 44).

In the Documentation Updates section:

- Deleted the corrections for the *Intel Dialogic System Release 6.0 cPCI Feature Pack 1 for Windows Software Installation Guide*, because these corrections have been incorporated into the updated document that is now on the online documentation bookshelf.
- Added PTR# 36278 under *Intel Dialogic System Software for DM3 Architecture Products on Windows Diagnostics Guide*.
- Added PTR# 36260 under *Native Configuration Management API Library Reference*.
- Added PTR# 36726 under *Global Call E1/T1 CAS/R2 Technology User's Guide*.
- Added PTR# 35565 under *Modular Station Interface API Library Reference*.

Document Rev 13 - published December 23, 2005

Updated for Service Update 47.

In the Post-Release Developments, added Dynamic Selection of Signaling Type for a Trunk.

In the Release Issues section, added the following resolved problems: 33067, 36245, 36371

In the Documentation Updates section, added documentation updates to the following documents because of a new feature in the Service Update: *Intel NetStructure for DM3 Architecture for cPCI on Windows Configuration Guide*, *Global Call API Library Reference*, *Global Call ISDN Technology Guide*, and DCM Online Help

Document Rev 12 - published December 6, 2005

Updated for Service Update 45.

In the Release Issues section, added the following resolved problem: 36318

Document Rev 11 - published December 2, 2005

Updated for Service Update 44.

In the Post-Release Developments section:

- Added Retrieving Coder Information from Fast Start Call Offers.
- Added Extended H.323 Nonstandard Data. Related to this feature, also made some changes to the Using H.323 Annex M Tunneled Signaling Messages section.

In the Release Issues section:

- Added the following resolved problems: 36307, 36310

- Added the following known (permanent) problem: 34616

In the Documentation Updates section, added documentation updates to the *Global Call IP Technology Guide* because of new features in the Service Update.

Document Rev 10 - published November 29, 2005

Updated for Service Update 42.

In the Release Issues section, added the following resolved problem: 33496

In the Documentation Updates section, added PTR# 36373 under *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide*.

Document Rev 09 - published November 4, 2005

Updated for Service Update 41.

In the Post-Release Developments section, added ISDN Network Side Conformance to Network Protocol Standards ITU-T Q921 and Q931.

In the Release Issues section:

- Added the following resolved problem: 36085
- Changed PTR 32087 to a known (permanent) issue, which has been added as a documentation update for the *Global Call IP Technology Guide.*

In the Documentation Updates section, added documentation updates to the following documents because of a new feature in the Service Update: *Intel NetStructure for DM3 Architecture for cPCI on Windows Configuration Guide* and *Global Call ISDN Technology Guide*

Document Rev 08 - published October 21, 2005

Updated for Service Update 39.

In the Post-Release Developments section, added support for Redundant Host (RH) and Redundant System Slot (RSS) on Intel NetStructure® ZT5084 with ZT5550C SBC under RH, RSS, and PHS Support on Additional Compute Platforms.

In the Release Issues section, added the following resolved problems: 32919, 35440, 35776, 35954, 36083

In the Documentation Updates section:

- Under *Intel Dialogic System Release 6.0 cPCI Feature Pack 1 for Windows Release Guide*:
  - Updated the information about Redundant Host (RH) and Redundant System Slot (RSS) to include Intel NetStructure® ZT5084 with ZT5550C SBC.

  - Added PTR# 36031.
- Corrected the order of procedures in the *Intel Dialogic System Release 6.0 cPCI Feature Pack 1 for Windows Software Installation Guide*.
- Added PTR# 35769 under *Intel NetStructure for DM3 Architecture for cPCI on Windows Configuration Guide*.
- Added PTR# 34237 under *Global Call API Library Reference*.
- Added PTR# 32087 under *Global Call IP Technology Guide*. Also clarified some of the field descriptions for the IP_VIRTBOARD data structure.
- Added PTR# 33826 under *IP Media Library API Programming Guide*.
- Added PTR# 32966 under *Standard Runtime Library API Programming Guide* and *Voice API Programming Guide*.

Document Rev 07 - published September 23, 2005

Updated for Service Update 37.

In the Release Issues section:

- Added the following resolved problems: 35101, 35520, 35585, 35596, 35693, 35696, 35797
- Added the following known (permanent) problems: 33991, 36016

In the Documentation Updates section, added PTR# 33975 and 35747 under *Intel Dialogic System Software for DM3 Architecture Products on Windows Diagnostics Guide*.

Document Rev 06 - published August 26, 2005

Updated for Service Update 36.

In the Post-Release Developments section, added Support for QSIG NCAS on DM3 Boards.

In the Release Issues section:

- Added the following resolved problems: 35197, 35250, 35507, 35704, 35705
- Added a known problem (no PTR number) with the host install affecting the use of PDKManager after an update install.
- Added a known problem (no PTR number) with the high availability demo.

In the Documentation Updates section, under *Intel Dialogic System Release 6.0 cPCI Feature Pack 1 for Windows Release Guide*, updated name of ZT5085 chassis to MPCHC5085.

Document Rev 05 - published July 1, 2005

Updated for Service Update 33.

In the Post-Release Developments section, added New Features in Global Call Protocols Package.

In the Release Issues section, added the following resolved problems: 28892, 30233, 32384, 33667, 34274, 34319, 34329, 34537, 34543, 34586, 34587, 34663, 34664, 34685, 34878, 34945, 34972, 34985, 35035, 35042, 35049, 35157, 35159, 35188, 35321

In the Documentation Updates section:

- Added PTR# 35263 under *Intel Dialogic System Release 6.0 cPCI Feature Pack 1 for Windows Software Installation Guide*.
- Added PTR# 33852 and made a correction to the GCLIB_MAKECALL_BLK data structure reference page under *Global Call API Library Reference*.

Document Rev 04 - published May 13, 2005

Updated for Service Update 28.

In the Post-Release Developments section:

- Added Windows Server 2003 SP1 Support.
- Added Enhancement to FEATURE_TABLE Data Structure for FSK Support.
- Added ISDN Trace Capability on Multiple Trunks.
- Added Intel NetStructure® MPCHC5091 to the PHS Support on Additional Compute Platforms section.
- Added GCEV_TRACEDATA Reference to the Tracing CAS Signaling section.

In the Release Issues section, added the following resolved problems: 28628, 33351, 33385, 33492, 33685, 34310, 34503, 34874, 34988, 35051, 35106, 35110, 35138, 35140

In the Documentation Updates section:

- Added documentation updates to the following documents because of new features in the Service Update: *Intel Dialogic System Release 6.0 cPCI Feature Pack 1 for Windows Release Guide*, *Intel Dialogic System Software for DM3 Architecture Products on Windows Diagnostics Guide*, *Voice API Library Reference*
- Added PTR# 29448 to *Global Call E1/T1 CAS/R2 Technology User's Guide*.

Document Rev 03 - published April 15, 2005

Updated for Service Update 18.

In the Post-Release Developments section, added the following new features:

- PHS Support on Additional Compute Platforms
- Mixing ISDN and Clear Channel on a DMN160TEC or DMT160TEC Board
- Using H.323 Annex M Tunneled Signaling Messages

In the Release Issues section:

- Added the following resolved problems: 31146, 32005, 33011, 33604, 33687, 34050, 34505, 34848, 34881, 34891
- Added “SU No.” column to the Issues table to show the Service Update number for resolved PTRs. Also added a link to view the Issues table sorted by Service Update number.

In the Documentation Updates section, added documentation updates to the following documents because of new features in the Service Update: *Intel Dialogic System Release 6.0 cPCI Feature Pack 1 for Windows Release Guide*, *Intel Dialogic System Release 6.0 cPCI Feature Pack 1 for Windows Software Installation Guide*, *Intel NetStructure for DM3 Architecture for cPCI on Windows Configuration Guide*, *Global Call IP Technology Guide*, DCM Online Help

Document Rev 02 - published February 9, 2005

Updated for Service Update 8.

Added a new section, Post-Release Developments, to describe the new features provided in the Service Update.

Added the following resolved problems to the Release Issues section: 32918, 33156, 33556, 33686, 33778, 33823, 34038

Added a known problem for Host Install (regarding board auto detect) to the Release Issues section.

In the Documentation Updates section:

- Added documentation updates to the following documents because of new features provided in the Service Update: *Intel Dialogic System Release cPCI Feature Pack 1 for Windows Release Guide*, *Intel NetStructure for DM3 Architecture for cPCI on Windows Configuration Guide*, *Native Configuration Management API Library Reference*, *Intel Dialogic System Software for DM3 Architecture Products on Windows Diagnostics Guide*, *Audio Conferencing API Library Reference*, *Global Call API Library Reference*
- Added information about diagnosing communication link errors between an HDSI Board and SIB to the *Intel Dialogic System Software for DM3 Architecture Products on Windows Diagnostics Guide*.

Document Rev 01 - published September 2004

Initial Version of document.

Post-Release Developments

This section describes significant changes to the system release subsequent to the general availability release date.

1.1 Service Update for System Release 6.0 CompactPCI Feature Pack 1

A Service Update for System Release 6.0 CompactPCI Feature Pack 1 for Windows is now available. Service Updates provide fixes to known problems, and may also introduce new functionality. New versions of the Service Update are planned to be released periodically. It is intended that this Release Update will document the features in the Service Updates.

Depending on whether you already have a version of System Release 6.0 CompactPCI Feature Pack 1 on your system, installing the Service Update will give you either a **full install** or an **update install**:

- If you don't have an existing version of System Release 6.0 CompactPCI Feature Pack 1 on your system, installing the Service Update gives you a **full install** of the release. You can select the features that you want to install, for example, Intel NetStructure DMV/DMN/DMT, Intel NetStructure DMIP, SNMP Agent Software, etc.

  Note: With the Service Update, the Global Call Protocols Package can now be installed as part of System Release 6.0 CompactPCI Feature Pack 1. Previously, this package was installed separately.

- If you have an existing version of System Release 6.0 CompactPCI Feature Pack 1 on your system, installing the Service Update gives you an **update install**. The update install gives you the latest software for the features that you selected when you did the full install of the system release that is currently on your system. If you want additional features, you can use the Modify or Change option as explained in the Installation Guide.

Note: Since the Global Call Protocols Package is now included with this Service Update version of System Release 6.0 CompactPCI Feature Pack 1, the stand-alone protocols package must not be used. (If you already have the stand-alone protocols package installed, you will be prompted to remove it before installing the Service Update.) Do not install the stand-alone protocols package after installing the Service Update (full install or update install), or your software may become non-functional.

See the new *Intel Dialogic System Release 6.0 CompactPCI Feature Pack 1 for Windows Service Update Software Installation Guide* on the online documentation bookshelf for complete, detailed information about installing the software.

1.2 Global Call Support for Time Slots on SS7 Boards Running in DTI Mode

With the Service Update, Global Call works with SS7 boards that include trunks not configured for SS7 signalling (DTI mode); i.e., all the time slots on these trunks operate in clear channel mode.

1.2.1 Feature Description

The SS7 boards are supported in a non SS7 signaling environment. This allows the application to use an SS7 board (for example, SS7HDCN16) in a clear channel mode to terminate E1/T1 trunks and switch them over the CT bus.

1.2.2 Supported Boards

- Intel NetStructure® SS7HDCN16
- Intel NetStructure® SS7HDCS8
- Intel NetStructure® SS7HDCD16
- Intel NetStructure® SS7HDCQ16

1.2.3 Configuration

The following three files must be configured to use this feature:

- *gcss7.cfg*
- *config.txt*
- *system.txt*

1.2.3.1 Global Call SS7 Software Configuration (gcss7.cfg)

The Global Call software configuration file (*gcss7.cfg*) is modified to specify a new **ClearGrp** parameter that includes the following fields:

<trunk_name>

The virtual device name of the trunk (for example, dkB1)

Note: This specifies the physical device where the circuits in the group are terminated. The <trunk_name> refers to one of the trunks on an Intel NetStructure SS7 board, where dkB1 is the name of the first trunk (first LIU defined in the *config.txt* file), dkB2 is the name of the second trunk (second LIU defined in the *config.txt* file), and so on. The same name is used as a basis by the application for the network device name when it opens a Global Call SS7 device.

<ts_mask>

Specifies the time slots that are to be used as clear channels. Each bit in this mask corresponds to a physical time slot on the trunk where a “1” indicates that the time slot

is to be used in clear channel mode. For example, 0x7fffffff for E1 or 0xffffff for T1 indicates that all time slots in the trunk are to be used in clear channel mode.

For example:

```
# Clear Channel Group configuration."
# ClearGrp <"trunk_name"> <ts_mask>
ClearGrp dkB1 0x7fffffff
ClearGrp dkB2 0x7fffffff
ClearGrp dkB3 0x7fffffff
ClearGrp dkB4 0x7fffffff
```

1.2.3.2 config.txt

In clear channel mode, the minimum configuration required is the board and LIU configuration in the *config.txt* file, as shown in the example below for an SS7HDC board:

```
* For SS7HD cP boards:
* SS7_BOARD <board_id> <board_type> <flags> <code_file> <run_mode>
SS7_BOARD  0 SS7HDC 0x00C2  ss7.dc4  dti
*
* LIU_CONFIG <board_id> <liu_id> <liu_type> <line_code> <frame_format> <crc_mode>
LIU_CONFIG 0 0 5 1 1 1
LIU_CONFIG 0 1 5 1 1 1
LIU_CONFIG 0 2 5 1 1 1
LIU_CONFIG 0 3 5 1 1 1
```

1.2.3.3 system.txt

The following is an example of the minimum configuration required in the *system.txt* file for clear channel operation (i.e, when there is no SS7 signaling supported):

```
*
* Essential modules running on host:
*
LOCAL           0x20            * ssd/ssds/ssdh - Board interface task
LOCAL           0x00            * tim_nt - Timer task
*
* Optional modules running on the host:
*
LOCAL           0xcf            * s7_mgt - Management/config task
LOCAL           0xef            * s7_log - Display and logging utility
LOCAL           0x4d            * GCSS7
*
*
* Essential modules running on the board (all redirected via ssd):
*
REDIRECT        0x10    0x20    * CT bus/Clocking control module
REDIRECT        0x8e    0x20    * On-board management module
*
*


* Redirection of status indications:
*
*REDIRECT       0xdf    0xef    * LIU/MTP2 status messages -> s7_log
*REDIRECT        0xdf    0x4d    * LIU/MTP2 status messages -> GCSS7
*REDIRECT       0xef    0x4d    * trace messages -> GCSS7
*
* Now start-up all local tasks:
*   (For PCCS6 start-up use ssd.exe and ssd_poll.exe,
*   for SPCI4/SPCI2S/CPM8 start-up use ssds.exe and
```

```
*   for SS7HD boards use ssdh.exe)
*
* FORK_PROCESS  ssd.exe
* FORK_PROCESS  ssd_poll.exe
* FORK_PROCESS  ssdh.exe
FORK_PROCESS    ./ssds -d
FORK_PROCESS    ./tim_lnx
FORK_PROCESS    ./tick_lnx
FORK_PROCESS    ./s7_mgt -d -i0x4d
FORK_PROCESS    ./s7_log
* FORK_PROCESS  upe.exe
```

1.2.3.4 Additional Configurations

In addition to the configuration for clear channel (DTI mode), there can be additional configurations for SS7 signaling on other SS7 boards in the system. The following are examples of a mixed configuration (SS7 signaling on a SS7HDCS8 board and clear channels on SS7HDCN16 board):

```
*********************************************************************
* Example Protocol Configuration File (config.txt) for use with
* Intel(R) NetStructure(TM) SS7 Boards.
*
* Boards supported are PCCS6, SPCI4, SPC2S, CPM8 and the SS7HD range.
* (note, not all boards are supported on all operating systems).
*
* This file needs to be modified to suit individual circumstances.
* Refer to the relevant Programmer's Manuals for further details.
*
*****************************************************************************
* For SS7HD cP boards:
* SS7_BOARD <board_id> <board_type> <flags> <code_file> <run_mode>
SS7_BOARD  0 SS7HDC 0x0042  ss7.dc4  ISUP * Master -- HDC
SS7_BOARD  1 SS7HDC 0x00C2  ss7.dc4  dti

* Configure individual E1/T1 interfaces:
* LIU_CONFIG <board_id> <liu_id> <liu_type> <line_code> <frame_format>
*            <crc_mode>
LIU_CONFIG  0  0  4  4  7  4
LIU_CONFIG  0  1  4  4  7  4
LIU_CONFIG  0  2  4  4  7  4
LIU_CONFIG  0  3  4  4  7  4
LIU_CONFIG  1  0  4  4  7  4
LIU_CONFIG  1  1  4  4  7  4
LIU_CONFIG  1  2  4  4  7  4
LIU_CONFIG  1  3  4  4  7  4
LIU_CONFIG  1  4  4  4  7  4
LIU_CONFIG  1  5  4  4  7  4
LIU_CONFIG  1  6  4  4  7  4
LIU_CONFIG  1  7  4  4  7  4
LIU_CONFIG  1  8  4  4  7  4
LIU_CONFIG  1  9  4  4  7  4
LIU_CONFIG  1  10  4  4  7  4
LIU_CONFIG  1  11  4  4  7  4
LIU_CONFIG  1  12  4  4  7  4
LIU_CONFIG  1  13  4  4  7  4
LIU_CONFIG  1  14  4  4  7  4
LIU_CONFIG  1  15  4  4  7  4

* MTP_CONFIG <reserved> <reserved> <options>
MTP_CONFIG  0  0  0x00040f00
```

```
* MTP_LINKSET <linkset_id> <adjacent_spc> <num_links> <flags> <local_spc> <ssf>
MTP_LINKSET  0  777  1  0x0000  555  0x03 * Loopback - HDC 1
MTP_LINKSET  1  555  1  0x0000  777  0x03 * Loopback - HDC 2

* MTP_LINK <link_id> <linkset_id> <link_ref> <slc> <board_id> <blink>
*          <stream> <timeslot> <flags>
* Note 1: For PCCS6 boards the first LIU port is stream=16 whilst for other
* boards the first LIU port is stream=0.
* Note 2: The SS7HD board requires a compound parameter for blink of the form
* sp_id-sp_channel.
*
* For SS7HD boards:
MTP_LINK  0  0  0  0  0  0-0  0   24  0x0006 * LIU 0, signalling on TS 24, for loopback
MTP_LINK  1  1  0  0  0  0-1  1   24  0x0006 * LIU 1, signalling on TS 24, for loopback
*
* Define a route for each remote signaling point:
* MTP_ROUTE <dpc> <linkset_id> <user_part_mask>
MTP_ROUTE  777  0  0x0020  0x0000  0
MTP_ROUTE  555  1  0x0020  0x0000  0
*

* ISUP parameters:
*
* Configure ISUP module:
* ISUP_CONFIG <reserved> <reserved> <user_id> <options> <num_grps> <num_ccts>
ISUP_CONFIG  0  0  0x4d  0x0774  2  50

*
* Configure ISUP circuit groups:
* ISUP_CFG_CCTGRP <gid> <dpc> <base_cic> <base_cid> <cic_mask> <options>
*                       <user_inst> <user_id> <opc> <ssf> <variant> <options2>
ISUP_CFG_CCTGRP  0  777  0x01  0x00  0x7fffff 0x0000401c  0  0x4d  555  0x3  2  0x00
ISUP_CFG_CCTGRP  1  555  0x01  0x19  0x7fffff 0x0000401c  0  0x4d  777  0x3  2  0x00



**********************************************************************
*
* Example System Configuration File (system.txt) for use with the
* Windows Development Package for Intel(R) NetStructure(TM) SS7 Boards
*
* Edit this file to reflect your configuration.
*
**********************************************************************
*
* Essential modules running on host:
*
LOCAL           0x20            * ssd/ssds/ssdh - Board interface task
LOCAL0x00* tim_nt - Timer task
*
* Optional modules running on the host:
*
LOCAL0xcf* s7_mgt - Management/config task
LOCAL           0xef            * s7_log - Display and logging utility

*LOCAL0x2d* upe - Example user part task
LOCAL0x4d* GC SS7 Service
*
* Modules that optionally run on the host
*
*LOCAL 0x23* ISUP module
*LOCAL 0x4a* TUP module
*LOCAL 0x33* SCCP module
```

```
*LOCAL 0x14* TCAP module
*LOCAL0x22* MTP3 module
*
* Essential modules running on the board (all redirected via ssd):
*

REDIRECT0x710x20* MTP2 module (except SS7HD boards)
REDIRECT0x81    0x20    * MTP2 module_id for SP 0 (SS7HD boards only)
REDIRECT0x91    0x20    * MTP2 module_id for SP 1 (SS7HD boards only)
REDIRECT0xe1    0x20    * MTP2 module_id for SP 2 (SS7HD boards only)
REDIRECT0xf1    0x20    * MTP2 module_id for SP 3 (SS7HD boards only)
REDIRECT        0x10    0x20    * CT bus/Clocking control module
REDIRECT0x8e0x20* On-board management module
*
* Modules that optionally run on the board (all redirected via ssd):
*

REDIRECT 0x230x20* ISUP module
* REDIRECT 0x4a0x20* TUP module
* REDIRECT 0x330x20* SCCP module
* REDIRECT 0x140x20* TCAP module
REDIRECT0x220x20* MTP3 module
*
* Redirection of status indications:
*
REDIRECT0xdf0x4d* LIU/MTP2 status messages -> GCSS7
*REDIRECT0xef0x4d* other/trace -> GCSS7
REDIRECT0xdf0xef* LIU/MTP2 status messages -> s7_log
*
* Now start-up all local tasks:
*   (For PCCS6 start-up use ssd.exe and ssd_poll.exe,
*   for SPCI4/SPCI2S/CPM8 start-up use ssds.exe and
*   for SS7HD boards use ssdh.exe)
*
* FORK_PROCESSssd.exe
* FORK_PROCESSssd_poll.exe
* FORK_PROCESSssds.exe
FORK_PROCESSssdh.exe -d
FORK_PROCESSStim_nt.exe
FORK_PROCESSStick_nt.exe
FORK_PROCESSs7_mgt.exe -d -i0x4d
FORK_PROCESSs7_log.exe
* FORK_PROCESSupe.exe
*
*
*****************************************************************************
```

1.2.4 Feature Limitations, Caveats and Restrictions

- The following APIs are supported in clear channel mode:
  - **gc_Attach( )**
  - **gc_AttachResource( )**
  - **gc_Close( )**
  - **gc_Detach( )**
  - **gc_GetNetworkH( )**
  - **gc_GetResourceH( )**
  - **gc_GetXmitSlot( )**
  - **gc_Listen( )**
  - **gc_OpenEx( )**

- **gc_Start( )**
- **gc_Stop( )**
- **gc_UnListen( )**
- **gc_GetVoiceH( )**

- None of the call control related APIs are supported for clear channel trunks (DTI mode) (for example, **gc_MakeCall( )**, **gc_WaitCall( )**, **gc_AnswerCall( )**, etc.).
- For clear channel circuits, if a call control function is issued, an error message is generated indicating that the API is not supported. The error value EGC_UNSUPPORTED is the Global Call value returned when the **gc_ErrorInfo( )** function is used to retrieve the error code.

1.2.5 Documentation

The online bookshelf provided with System Release 6.0 CompactPCI Feature Pack 1 for Windows contains information about all system release features including features for application development, configuration, administration, and diagnostics.

For more information about the Global Call API in general, see the following documents:

- *Global Call API Programming Guide*
- *Global Call API Library Reference*

For features specific to SS7 technology, see:

- *Global Call SS7 Technology Guide*

The online bookshelf has not been updated for this feature, so the *Global Call SS7 Technology Guide* does not currently indicate that SS7 boards are supported in a non SS7 signaling environment.

1.3 Windows Services Control Panel Name Changes

With the Service Update, changes have been to the Windows Services Control Panel, as follows:

- The Dialogic System Services is now Intel Dialogic product System Service.
- The DLGC Device Mapper has been removed as a service.

Note: To access the Services area, click the **Start** button on the Main Menu, and then select Control Panel->Administrative Tools->Services.

1.4 Notification of Layer 1 Alarm Events on SS7 Boards

With the Service Update, the support for alarm notification has been added for SS7 boards. By adding support for alarm notification, applications are able to better determine which devices are available for making and receiving calls, or enabling/disabling voice activity.

1.4.1 Feature Description

The application is notified whenever a MVD_MSG_LIU_STATUS message is received for SS7 boards. This message provides specific layer 1 status indications (for example, Pulse Code Modulation (PCM) Loss of Signal of Signal (LOS), Alarm Indication Signal (AIS), etc.) for dkBx trunk devices. This notification to the application is sent for each circuit associated with the trunk in the conventional Global Call method using GCEV_ALARM events.

The GCSS7 Library has been enhanced to support the Global Call Alarm Management System (GCAMS). This provides applications with notification when layer 1 alarms are present and when the alarms have cleared via GCEV_ALARM events. Applications have the ability to control the following:

- Which alarms are blocking and non-blocking.
- Alarm flow (for example, notification of when the first alarm occurred and the last alarm cleared) via **gc_SetAlarmFlow( )** API.

1.4.2 Supported Boards

Intel NetStructure® SS7HDC (CompactPCI) boards.

1.4.3 GCAMS Support

This section describes the alarms that are received from the SS7 stack. The alarms are propagated to the application via the GCEV_ALARM event. A new GCSS7 Alarm Source Object (ASO), which is a module in GCAMS that handles the SS7 layer 1 alarms, and function table(s) have been created to support this functionality.

1.4.3.1 GCSS7 ASO

The new GCSS7 ASO resides in the GCSS7 Call Control Library for the SS7 boards and pertains to SS7 layer 1 alarms only. The ASO handles the Line Interface Unit (LIU) indications from the underlying stack and the alarm notification.

1.4.3.2 Alarm Handling for SS7 Boards

When using SS7 boards, alarms are recognized on a span (trunk) basis. Once an alarm is detected, all open channels on that span receive a GCEV_BLOCKED event. When the alarm is cleared, open channels receive a GCEV_UNBLOCKED event. Alarm notification

(GCEV_ALARM event) is disabled by default and must be enabled via **gc_SetAlarmConfiguration( )**. When alarm notification is enabled, alarms (GCEV_ALARM events) are generated for each time slot on that affected span. See the *Global Call API Programming Guide* for more information.

1.4.3.3 Supported Alarms

The following list shows the alarms that are supported for SS7 boards. The dagger symbol (†) next to an alarm name indicates that the alarm is blocking by default.

The default can be changed using the **gc_SetAlarmConfiguration( )** function.

†SS7_FRAME_SYNC_LOSS (0xa)
Frame Sync Loss has been detected

†SS7_FRAME_SYNC_OK (0xb)
†Frame Sync has been cleared

SS7_AIS_DETECTED (0xc)
Alarm Indication Signal (AIS) has been detected

†SS7_AIS_CLEARED (0xd)
AIS has been cleared

†SS7_REMOTE_ALARM_DETECTED (0xe)
Remote Alarm has been detected

†SS7_REMOTE_ALARM_CLEARED (0xf)
Remote Alarm has been cleared

†SS7_PCM_LOSS (0x14)
Pulse Code Modulation (PCM) Loss has been detected

†SS7_PCM_OK (0x15)
PCM Loss has been cleared

SS7_FRAME_SLIP (0x16)
Frame Slip has been detected

†SS7_BER5 (0x19)
Bit Error Rate (BER) > 1 in 100,000 has been detected

†SS7_BER5_CLEARED (0x1a)
BER5 has been cleared

†SS7_BER3 (0x1b)
BER > 1 in 1,000 has been detected

†SS7_BER3_CLEARED (0x1c)
BER3 has been cleared

1.4.3.4 SS7-Specific Event Cause Codes for Layer 1 Alarms

When an event is received, the **gc_ResultInfo( )** function, or the **gc_ResultValue( )** function (deprecated), can be used to retrieve event cause code information.

- When the **gc_ResultInfo( )** function is used, the **a_Info** parameter is a pointer to a GC_INFO structure that contains both the standard Global Call event cause code (gcValue field), and an SS7-specific event cause code (ccValue field).
- When the **gc_ResultValue( )** function is used, function parameters point to a standard Global Call event cause code (**gc_resultp** function parameter), and an SS7-specific event cause code (**cclib_resultp** function parameter).

The SS7-specific event cause codes related to layer 1 alarms are:

S7RV_PCM_LOSS (0x40a9)
Loss of PCM stream

S7RV_PCM_OK (0x40aa)
Recovery of PCM stream

S7RV_FRAME_SYNC_LOSS (0x40ab)
Loss of frame synchronization

S7RV_FRAME_SYNC_OK (0x40ac)
Recovery of frame synchronization

S7RV_AIS_DETECTED (0x40ad)
Alarm Indication Signal (AIS) alarm detected

S7RV_AIS_CLEARED (0x40ae)
Alarm Indication Signal (AIS) alarm cleared

S7RV_FRAME_SLIP (0x40af)
Frame Slip detected

S7RV_BER3 (0x40b0)
A Bit Error Rate (BER) of greater than 1 in 1,000 has been detected

S7RV_BER3_CLEARED (0x40b1)
A BER of greater than 1 in 1,000 has been cleared

S7RV_BER5 (0x40b2)
A BER of greater than 1 in 100,000 has been detected

S7RV_BER5_CLEARED (0x40b3)
A BER of greater than 1 in 100,000 has been cleared

S7RV_REMOTE_ALARM_DETECTED (0x40b4)
Detection of remote alarm

S7RV_REMOTE_ALARM_CLEARED (0x40b5)
Clearing of remote alarm

1.4.4 Example

1.4.4.1 Enable reception of GCEV_ALARM Event

```
int EnableAlarmEvent()
{
          int rc = gc_OpenEx();

          if(rc != GC_SUCCESS) {
                    cout << "failed to open device" << endl;
                    return GC_ERROR;
          } else {
                    Cout << "gc_OpenEx() called - device successfully opened" << endl;
          }
          rc = gc_SetAlarmNotifyAll();

          if(rc != GC_SUCCESS) {
                    cout << "failed to enable reception of GCEV_ALARM event" << endl;
                    return GC_ERROR;
          } else {
                    Cout << "gc_SetAlarmNotifyAll() called - successfully enabled reception of
                    GCEV_ALARM event" << endl;
          }
          return 0;
}
```

1.4.4.2 Processing of GCEV_ALARM Event

```
int ProcessGCEvent(METAEVENT *metaeventp)
{
          Char alarm_name[30];
          switch (metaeventp->evttype)
          {
                    :
                    case GCEV_ALARM:

                              gc_AlarmNumberToName(ALARM_SOURCE_ID_SS7, alarm_number,
                              &alarm_name);

                              cout << "Received GCEV_ALARM because of " << alarm_name << endl;
                    break;
                    :
          }

          return 0;

}
```

1.4.5 Feature Limitations, Caveats and Restrictions

The set of Global Call functions that comprise the GCAMS interface for alarm management on SS7 boards are supported with the following restrictions:

- Using GCAMS, the application has the ability to set which alarms are blocking and nonblocking as described in the *Global Call API Programming Guide*. However, this capability applies on a span basis only. Changing which alarms are blocking and non-blocking for one time slot results in changing which alarms are blocking and non-blocking for all time slots on the span. This is because all the time slots on the span

use the same ASO and once you change the ASO configuration it affects all the time slots.

- For SS7 technology, the **gc_TransmitAlarms( )** and **gc_StopTransmitAlarms( )** functions are not supported on SS7 boards since there are no interfaces in the underlying stack to support transmission of layer 1 alarms.
- Using the **gc_GetAlarmParm( )** and **gc_SetAlarmParm( )** functions to retrieve and set specific alarm parameters, for example alarm triggers, is not supported.

1.4.6 Documentation

The online bookshelf provided with System Release 6.0 CompactPCI Feature Pack 1 for Windows contains information about all system release features including features for application development, configuration, administration, and diagnostics.

For more information about the Global Call API in general, including GCAMS, see the following documents:

- *Global Call API Programming Guide*
- *Global Call API Library Reference*

For features specific to SS7 technology, see:

- *Global Call SS7 Technology Guide*

The online bookshelf has not been updated for this feature, so the *Global Call SS7 Technology Guide* does not currently indicate that layer 1 alarms are supported for SS7 boards.

1.5 Automatic FCD File Generation

With the Service Update, the fcdgen utility is no longer required to generate the FCD file. When you download a PCD file and its corresponding CONFIG file to a board, the FCD file is automatically generated and also downloaded to the board. The FCD file is also copied into the data directory.

With this enhancement to the configuration process, it is no longer necessary to use the fcdgen utility to generate a modified FCD file. When you modify a CONFIG file, the modified FCD file is automatically created when the PCD file and CONFIG file are downloaded to the board.

1.5.1 Documentation

The online bookshelf provided with System Release 6.0 CompactPCI Feature Pack 1 for Windows contains information about all system release features including features for application development, configuration, administration, and diagnostics.

For detailed information about configuring DM3 boards, see the *Intel NetStructure for DM3 Architecture for CompactPCI on Windows Configuration Guide*.

Note: The online bookshelf has not been updated for this feature, so the Configuration Guide does not currently include information about automatic FCD file generation.

1.6 New OAMIPC Mechanism Replaces CORBA

With the Service Update, a new OAMIPC mechanism replaces CORBA and CORBA will no longer be used. This mechanism changes the binary size of the oam binaries. The ooc directory under the dialogic directory will be removed if you are doing an upgrade install, or the ooc directory will not be installed in case of a new installation.

As part of the new OAMIPC, the TCP Port List for new installation and upgrade installation has changed. For new installations, you will see the Welcome screen with the message "Exclusive access to TCP ports 12001, 12004-5 for the loopback interface, and port 12002 for all network interfaces is required so ensure that these ports are available on your system."

If you are performing an upgrade installation, you will not see the message, but you will still have to check that these TCP ports are available before you perform the upgrade.

Refer to the *Intel Dialogic System Release 6.0 CompactPCI Feature Pack 1 for Windows Service Update Software Installation Guide* section about "Checking TCP Port Availability" for further information.

1.7 Enhanced Runtime Trace Facility (RTF)

The Service Update provides enhancements to Runtime Trace Facility (RTF) logging. The RTF tool provides a mechanism for tracing the execution path of Dialogic runtime libraries. The trace information can be captured in a log file or sent to a system-specific debug stream (e.g., debug console on Windows). The resulting log file/debug stream output helps troubleshoot runtime issues for applications that are built with Dialogic software.

For detailed information about RTF logging, see the Diagnostics Guide.

1.8 Intel Telecom Subsystem Summary Tool (its_sysinfo) Command Line Execution

The Intel Telecom Subsystem Summary Tool (its_sysinfo) provides a simple way to collect information about systems built using Intel telecom products. The its_sysinfo tool collects data from the system on which you execute it and provides you with information about the system environment: the operating system, computer architecture, Intel Dialogic System Release, and operational logs.

For detailed information about the its_sysinfo tool, including the new option for command line execution, see the Diagnostics Guide.

1.9 Call Rejection Notification for Blocked Channel Recovery

With applications using ISDN, problems such as the host library call states getting out of sync with the firmware call states could occur. This can cause the firmware to reject inbound calls on a particular channel, leaving the channel/time slot permanently blocked. After repeated channel failures, some switches place the trunk out-of-service. If the application is not notified of this condition, it could eventually be operating with reduced capacity.

With the Service Update, a new event will now notify the application when incoming calls are rejected and provide the reason for the rejection. Upon receipt of this call rejection notification event, the application can initiate blocked channel recovery procedures so the channel can accept calls again.

This feature is supported on Intel NetStructure® DM/V-A, DMN160TEC, and DMT160TEC boards for ISDN protocols only.

1.9.1 Feature Description

This feature allows the user to dynamically enable or disable a call rejection notification event on any trunk of a DM/V-A, DMN160TEC, or DMT160TEC board at runtime. The call rejection notification event is a means of alerting the application when firmware is rejecting calls on some channel of a trunk. It is up to the application to initiate blocked channel recovery procedures.

The unsolicited GCEV_EXTENSION event is used for this feature. The procedure for implementing this feature in an application includes:

- Enabling Notification of Incoming Call Rejection
- Processing the GCEV_EXTENSION Event
- Recovering the Blocked Channel

Enabling Notification of Incoming Call Rejection

The application can be notified of calls being rejected by the firmware via the GCEV_EXTENSION event. Information about the event is contained in the EXTENSIONEVTBLK structure, which is referenced via the extevtdatap pointer in the METAEVENT structure associated with the GCEV_EXTENSION event. This notification is disabled by default and must be enabled by issuing **gc_util_insert_parm_val( )** to build a GC_PARM_BLK, followed by **gc_SetConfigData( )** to enable the event. For example:

```
gc_util_insert_parm_val(&ParmBlkp, CCSET_EXTENSIONEVT_MSK, GCACT_ADDMSK, sizeof(int),
EXTENSIONEVT_CALL_REJECTED);
```

```
gc_SetConfigData(GCTGT_CCLIB_NETIF, trunk, ParmBlkp, 0, GCUPDATE_IMMEDIATE, &id, EV_ASYNC);
```

The parameter values for the **gc_util_insert_parm_val( )** function are:

- **parm_blkpp** = pointer to the address of a valid GC_PARM_BLK structure where the parameter and value are to be inserted
- **setID** = CCSET_EXTENSIONEVT_MSK
- **parmID** = GCACT_ADDMSK to enable the event, or GCACT_SUBMSK to disable the event
- **data_size** = sizeof(int)
- **data** = EXTENSIONEVT_CALL_REJECTED

Once the GC_PARM_BLK has been populated with the desired values, the **gc_SetConfigData( )** function can be issued to perform the configuration. The parameter values for the **gc_SetConfigData( )** function are:

- **target_type** = GCTGT_CCLIB_NETIF
- **target_id** = the trunk line device handle, as obtained from **gc_OpenEx( )** with a devicename string of ":N_dtiBx:P...".
- **target_datap** = GC_PARM_BLKP parameter pointer, as constructed by the utility function **gc_util_insert_parm_val( )**
- **time_out** = time interval (in seconds) during which the target object must be updated with the data. If the interval is exceeded, the update request is ignored. This parameter is supported in synchronous mode only, and it is ignored when set to 0.
- **update_cond** = GCUPDATE_IMMEDIATE
- **request_idp** = pointer to the location for storing the request ID
- **mode** = EV_ASYNC for asynchronous execution or EV_SYNC for synchronous execution

See the *Global Call API Library Reference* for further information about these functions.

Processing the GCEV_EXTENSION Event

The extension ID **DM3CC_EXT_EVT_CALL_REJECTED** identifies a GCEV_EXTENSION event as a call rejection notification event. The following {set_id, parm_id} pairs provide details about the call rejection:

```
{CCSET_CHANNEL, CCPARM_CHANNEL_ID},
{CCSET_CHANNEL, CCPARM_INTERFACE_ID},
{CCSET_CALLREJECTION_INFO, CCPARM_CALLREJECTION_CAUSE},
{CCSET_CALLREJECTION_INFO, CCPARM_CALLREJECTION_CAUSE_ALIAS}
```

This information is sent to the application when call rejection takes place. It is packaged inside the EXTENSIONEVTBLK structure that accompanies the GCEV_EXTENSION event. The information is accessed through a METAEVENT structure as shown in the following sample code. For information about call rejection causes, see the table "Network Cause Values When Using DM3 Boards" in the *Global Call ISDN Technology Guide*.

```
if (sr_waitevt(-1) >= 0)
{
     EXTENSIONEVTBLK *ext_id;
     METAEVENT meta;
     GC_PARM_BLKP parmblk_p;
     GC_PARM_DATAP parmdata_p;

     gc_GetMetaEvent(&meta);
     ext_id = (meta.extevtdatap)->ext_id;
     parmblk_p = ((EXTENSIONEVTBLK *) meta.extevtdatap)->parmblk;
     switch(sr_getevttype())
     {

          [...]

       case GCEV_EXTENSION:
          if (ext_id == DM3CC_EXT_EVT_CALL_REJECTED)
          {
               printf("Received Call Rejection event on device %s\n",
                    ATDV_NAMEP(sr_getevtdev()));
               while (NULL != (parmdata_p = gc_util_next_parm(&parmblk_p, NULL))
               {
                   unsigned char datasize = parmblk_p->value_size;
                   if (datasize == sizeof(char))
                   {
                       printf("Parameter Set ID = %d, Parm ID = %d, Value = %c\n",
                              parmdata_p->set_ID, parmdata_p->parmID,
                              (char) parmdata_p->value_buf);
                   }
                   else if (datasize == sizeof(int))
                   {
                       printf("Parameter Set ID = %d, Parm ID = %d, Value = %d\n",
                              parmdata_p->set_ID, parmdata_p->parmID,
                              (int) parmdata_p->value_buf);
                   }
               }
          }
          break;

          [...]

       default:
            printf("Received event 0x%x on device %s\n",
            sr_getevttype(), ATDV_NAMEP(sr_getevtdev()));
          break;
     }
}
```

Recovering the Blocked Channel

Depending on the call rejection cause associated with the GCEV_EXTENSION event, the application can initiate blocked channel recovery procedures such as using the **gc_ResetLineDev( )** function to reset the line device state. When used in asynchronous mode, the GCEV_RESETLINEDEV event indicates successful termination of the **gc_ResetLineDev( )** function. After receiving this event, the application must issue a new **gc_WaitCall( )** function to receive the next incoming call on the channel.

See the *Global Call API Library Reference* for further information about these functions.

1.9.2 Documentation

The online bookshelf provided with System Release 6.0 CompactPCI Feature Pack 1 for Windows contains information about all system release features including features for application development, configuration, administration, and diagnostics.

For more information about the Global Call API, see the following documents:

- *Global Call API Programming Guide*
- *Global Call API Library Reference*

For features specific to ISDN technology, see:

- *Global Call ISDN Technology Guide*

Note: The online bookshelf has not been updated for this feature, so the manuals above do not contain information about call rejection notification for blocked channel recovery.

1.10 Dynamic Selection of Signaling Type for a Trunk

With the Service Update, the ability to change the signaling type of any trunk from ISDN to clear channel or vice versa at runtime is supported. The boards that support this feature are:

- Intel NetStructure® DMN160TEC
- Intel NetStructure® DMT160TEC

Note: This feature is supported for E1 configurations only, not for T1.

1.10.1 Feature Description

This feature allows the user to dynamically change the signaling type of an individual trunk. The feature is disabled by default. Before the feature can be used, it must be enabled in one of the following ways:

- Setting the new **Dynamic_ISDN_CC** parameter on the Trunk Configuration property sheet in the configuration manager (DCM) to a value of “true”. (The default is “false”.)
- Using the Native Configuration Manager (NCM) API to set the new **Dynamic_ISDN_CC** parameter to a value of “true”. See the *Native Configuration Manager API Programming Guide* and the *Native Configuration Manager API Library Reference* for more information.

Notes:1. Setting the **Dynamic_ISDN_CC** parameter to “true” for T1 configurations is not permitted. DCM will not accept the change.

2. Enabling or disabling this feature using the NCM API should only be done at initialization time. If an attempt is made to enable or disable the feature after initialization, the API call succeeds and the parameter value is changed, but the feature will not function correctly since the appropriate configuration has not been established at initialization time (for example, the 31st device was not created for switching to clear channel).

The general procedure for using the feature is as follows:

1. It is the application's responsibility to handle all active calls on a trunk and terminate them if necessary. If switching between clear channel and ISDN, close the 31st channel. If the 31st channel is not closed, step 4 below will fail.

2. Optionally issue the **gc_ResetLineDev( )** function on all channels of the affected trunk to prevent any incoming calls from being accepted while the change is made.

3. Call the **gc_util_insert_parm_ref( )** function to change the new parameter (**CCPARM_SIGNALING_TYPE**) to clear channel or ISDN in the GC_PARM_BLK associated with the trunk. (Further information about this new parameter is given below.)

4. Call the **gc_SetConfigData( )** function to implement the change.

5. When switching from clear channel to ISDN, call **gc_util_insert_parm_ref( )** followed by **gc_SetConfigData( )** to specify the correct protocol if necessary.

6. When the GCEV_SETCONFIGDATA event is received, the application can start receiving calls (using **gc_WaitCall( )**) or making calls (using **gc_MakeCall( )**) again.

Dynamically changing the signaling type of a trunk at runtime has no effect on the PCD and CONFIG files. PCD and CONFIG files are created at initialization time based on the initial parameter values selected for each trunk. PCD and CONFIG files that have dynamic switching enabled are identified by the string "_dyn" appended to the file name.

When using E1 technology, the following should be noted:

- If clear channel is selected for a given trunk, all 31 devices are available to the application. If ISDN is selected, the 31st device is hidden from the application.
- When switching from ISDN to clear channel, the application must issue **gc_OpenEx( )** on the 31st channel before that channel can be used.
- The time slot mapping is as follows:
  - Time slots TS1 to TS15 map to dtiB1T1 to dtiB1T15.
  - Time slots TS17 to TS31 map to dtiB1T16 to dtiB1T30.
  - Time slot TS16 maps to dtiB1T31.

Restrictions and Limitations

The following restrictions and limitations apply:

- If active calls on a trunk are not terminated before the application requests a change of signaling type, the calls are dropped.
- When this feature is enabled, an additional CT Bus time slot is allocated for each trunk regardless of whether the trunk is using a clear channel or ISDN signaling type. This means that an additional 16 CT Bus time slots are consumed per board.
- This feature should **not** be used on a trunk that is running the NFAS D-channel. Changing the signaling type on an NFAS D-channel trunk results in the loss of the D-channel for all trunks in the NFAS group.

- For a trunk that is initially set for clear channel, if the trunk is being switched to ISDN for the first time, **a protocol must be explicitly selected**. If this is not done, subsequent **gc_MakeCall( )** calls may fail. If the trunk was previously set to use a specific ISDN protocol, that protocol is used if a new protocol is not specified.
- Dynamically switching between clear channel and ISDN and vice versa has no effect on other trunk parameters (for example, user/network mode etc.).
- Dynamically switching between the signaling type of a trunk exposes or hides the dtiB1T31 device on a trunk. The mapping of E1 time slots to dti devices should **not** be modified in the CONFIG file, as this will cause the wrong E1 time slot to be hidden.

New Parameters

The feature is implemented using the Global Call **gc_SetConfigData( )** function and the GCSET_LINE_CONFIG parameter set. A new CCPARM_SIGNALING_TYPE parameter ID has been defined to support the dynamic selection of signaling type for a trunk functionality.

The CCPARM_SIGNALING_TYPE parameter ID can take one of the following values:

CC_SIGNALING_TYPE_ISDN
: Identifies the signaling type as ISDN.

CC_SIGNALING_TYPE_CLEAR
: Identifies the signaling type as clear channel.

Generated Events

The **gc_SetConfigData( )** function is issued in asynchronous mode to achieve this functionality, and one of the following notification events is generated for the application:

GCEV_SETCONFIGDATA
: Indicates that the change of signaling type is a success.

GCEV_SETCONFIGDATAFAIL
: Indicates failure, for example, the firmware did not complete the change of signaling type successfully, or one of the error codes defined below was returned.

Error Codes

One of the following success or error codes is generated by the **gc_SetConfigData( )** function:

GC_SUCCESS
: Success. The signaling type change has been implemented.

EGC_INVDEVNAME
: Invalid device name. This includes all device name syntax errors, attempts to access the 31st channel while running ISDN, and attempts to invoke the feature on a board that does not support it.

EGC_PARM_VALUE_ERR
Parameter value error. The signaling type specified is not one of the two supported values described above.

EGC_NOT_ENABLED
Operation not enabled. In this context, the user is attempting to switch the signaling type when that functionality is not enabled. This is a *new* error code introduced with this feature.

EGC_PARM_UPDATEPERM_ERR
Parameter update not allowed. This occurs if the 31st channel is still open and the user tries to switch from a clear channel to an ISDN signaling type.

EGC_SYSTEM
System error. A general internal error has occurred, such as a device mapper crash.

1.10.2 Switching from ISDN to Clear Channel Scenario

In this scenario, a DMN160TEC or DMT160TEC board is configured to run one trunk with ISDN signaling and the user wishes to switch that trunk to use clear channel mode. The last channel (the 31st for E1 front-ends) must be made accessible to the application. Traffic on the other trunks must not be affected.

The pseudo code below shows how this can be achieved. Note the following:

- The pseudo code shown in this section is written according to the recommended programming model, that is, the asynchronous programming model, with omissions for brevity and legibility as noted.
- For the sake of brevity, error checking has been omitted from the pseudo code. In practice, all Global Call API calls should be checked for success or failure.

Note: The **gc_SetConfigData( )** function can be used on a board device to reconfigure the signaling type for a trunk. However, it is the application's responsibility to handle all active calls on the trunk, and terminate them if necessary. In addition, the **gc_ResetLineDev( )** function may be issued on all channels (time slots) prior to issuing **gc_SetConfigData( )** to prevent incoming calls. If there are any active calls present at the time the **gc_ResetLineDev( )** or **gc_SetConfigData( )** function is issued, they are gracefully terminated internally. The application does not receive GCEV_DISCONNECT events when calls are terminated in this manner.

1. Create the new configuration information that indicates a switch to clear channel:

```
gc_util_insert_parm_ref(&ParmBlkp, CCSET_LINE_CONFIG, CCPARM_SIGNALING_TYPE, sizeof(int),
       CC_SIGNALING_TYPE_CLEAR);
```

2. Establish the new configuration on the desired trunk:

```
gc_SetConfigData(GCTGT_CCLIB_NETIF, trunk, ParmBlkp, 0, GCUPDATE_IMMEDIATE, &id, EV_ASYNC);
```

3. Clean up the configuration:

```
gc_util_delete_parm_blk(ParmBlkp);
```

4. Perform event handling for the reconfiguration, regardless of whether the event indicates success or failure:

```
if (sr_waitevt(-1) >= 0)
{
    METAEVENT meta;
    gc_GetMetaEvent(&meta);
    switch(sr_getevttype())
    {
        case GCEV_SETCONFIGDATA:
            printf("Received event GCEV_SETCONFIGDATA(ReqID=%d) on device %s\n",
                ((GC_RTCM_EVTDATA *)(meta.evtdatap))->request_ID,
                ATDV_NAMEP(sr_getevtdev()));
            break;
        case GCEV_SETCONFIGDATA_FAIL:
            printf("Received event GCEV_SETCONFIGDATAFAIL(ReqID=%d) on
                device %s, Error=%s\n", ((GC_RTCM_EVTDATA*)(meta.evtdatap))->request_ID,
                ATDV_NAMEP (sr_getevtdev()),
                ((GC_RTCM_EVTDATA *) (meta.evtdatap))->additional_msg);
            break;
        default:
            printf("Received event 0x%x on device %s\n",
                sr_getevttype(), ATDV_NAMEP(sr_getevtdev()));
            break;
    }
}
gc_util_delete_parm_blk(ParmBlkp);
```

5. Finally, open the 31st channel:

```
gc_OpenEx(&ldev, ":N_dtiB1T31:P_ISDN", EV_ASYNC, (void*)usrattr);
```

1.10.3 Switching from Clear Channel to ISDN Scenario

In this scenario, a DMN160TEC or DMT160TEC board is configured to run one trunk in clear channel mode and the user wishes to switch that trunk to use ISDN signaling. The last channel (the 31st for E1 front-ends) must be closed and made inaccessible. Traffic on the other trunks remains unaffected.

The pseudo code below shows how this can be achieved. Note the following:

- The pseudo code shown in this section is written according to the recommended programming model, that is, the asynchronous programming model, with omissions for brevity and legibility as noted.
- For the sake of brevity, error checking has been omitted from the pseudo code. In practice, all Global Call API calls should be checked for success or failure.

1. Close the 31st line device:

```
gc_close(channel[31]->lineDevice);
```

2. Create the new configuration information that indicates a switch to ISDN:

```
gc_util_insert_parm_ref(&ParmBlkp, CCSET_LINE_CONFIG, CCPARM_SIGNALING_TYPE, sizeof(int),
    CC_SIGNALING_TYPE_ISDN);
```

3. Establish the new configuration on the desired trunk:

```
gc_SetConfigData(GCTGT_CCLIB_NETIF, trunk, ParmBlkp, 0, GCUPDATE_IMMEDIATE, &id, EV_ASYNC);
```

4. Choose a protocol for the trunk:

```
gc_util_insert_parm_ref(&ParmBlkp, GCSET_PROTOCOL, GCPARM_PROTOCOL_NAME,
    strlen(protocolName) +1, protocolName);
```

Note: If the trunk started with clear channel and is being switched to ISDN for the first time, the protocol must be set explicitly. If the trunk was ISDN previously, the trunk reverts to using the previously used protocol if a new protocol is not selected by the user.

5. Establish the selected protocol on the trunk if necessary:

```
gc_SetConfigData(GCTGT_CCLIB_NETIF, trunk, ParmBlkp, 0, GCUPDATE_IMMEDIATE, &id, EV_ASYNC);
```

6. Clean up the configuration:

```
gc_util_delete_parm_blk(ParmBlkp);
```

7. Perform the proper event handling for the reconfiguration, whether the event indicates success or failure:

```
if (sr_waitevt(-1) >= 0)
{
    METAEVENT meta;
    gc_GetMetaEvent(&meta);
    switch(sr_getevttype())
    {
        case GCEV_SETCONFIGDATA:
            printf("Received event GCEV_SETCONFIGDATA(ReqID=%d) on device %s\n",
                ((GC_RTCM_EVTDATA *)(meta.evtdatap))->request_ID,
                ATDV_NAMEP(sr_getevtdev()));
            break;
        case GCEV_SETCONFIGDATA_FAIL:
            printf("Received event GCEV_SETCONFIGDATAFAIL(ReqID=%d) on
                device %s, Error=%s\n", ((GC_RTCM_EVTDATA*)
                meta.evtdatap))->request_ID, ATDV_NAMEP
                (sr_getevtdev()), ((GC_RTCM_EVTDATA *) (meta.evtdatap))->additional_msg);
            break;
        default:
            printf("Received event 0x%x on device %s\n",
                sr_getevttype(), ATDV_NAMEP(sr_getevtdev()));
            break;
    }
}
```

1.10.4 Documentation

The online bookshelf provided with System Release 6.0 CompactPCI Feature Pack 1 for Windows contains information about all system release features including features for application development, configuration, administration, and diagnostics.

For more information about the Global Call API, see the following documents:

- *Global Call API Programming Guide*
- *Global Call API Library Reference*

For features specific to ISDN technology, see:

- *Global Call ISDN Technology Guide*

Note: The online bookshelf has not been updated for this feature, so the *Global Call ISDN Technology Guide* does not currently indicate that the dynamic selection of signaling type for a trunk is supported.

1.11 Retrieving Coder Information from Fast Start Call Offers

With the Service Update, the Global Call IP call control library supports the ability to retrieve coder information from fast start call offers.

Any call offer that is received can potentially contain “fast start” coder information, in the form of a Session Description Protocol (SDP) offer in an INVITE request when using SIP or a fastStart element in a Setup message when using H.323. The IP call control library handles any such fast start coder information internally to begin the coder negotiation process, but it may be useful to the application to access the offered coder information, as well. The call control library can be configured at start-up to provide application access to fast start coder information for SIP or H.323 or both. When this access is enabled and the library accepts a fast start offer, the extra data associated with the GCEV_OFFERED event that is sent to the application will contain one or more additional parameter elements to convey the coder information that was contained in the offer.

1.11.1 Enabling Access to Fast Start Coder Information

Application access to fast start coder information is a feature that can be disabled or enabled independently for the SIP and H.323 protocols at the time the **gc_Start( )** function is called.

The **INIT_IPCCLIB_START_DATA( )** and **INIT_IP_VIRTBOARD( )** functions, which must be called before the **gc_Start( )** function, populate the IPCCLIB_START_DATA and IP_VIRTBOARD structures, respectively, with default values. The default values of the sip_msginfo_mask and h323_msginfo_mask fields in the IP_VIRTBOARD structure disable all optional message information access features, including access to fast start coder information. The default values of these data structure fields must be overridden with appropriate values for each IPT board device on which access needs to be enabled. For each of the two message information mask fields, the value that the application sets will typically be an OR of two or more defined mask values as described in the reference page for IP_VIRTBOARD in the *Global Call IP Technology Guide*.

The defined mask values that are used to enable access to fast start coder information are:

IP_SIP_FASTSTART_CODERS_IN_OFFERED
: enables application access to coder information contained in SDP offers in SIP INVITE requests

IP_H323_FASTSTART_CODERS_IN_OFFERED
enables application access to coder information contained in fastStart elements in H.323 Setup messages

Note that it is not possible to toggle the fast start coder information access between enabled and disabled states without stopping and restarting the system via **gc_Stop( )** and **gc_Start( )**.

The following code snippet shows how an application might initialize two virtual boards to enable basic message information access and access to fast start coder information for both SIP and H.323 protocols.

```
.
.
.
INIT_IPCCLIB_START_DATA(&ipcclibstart, 2, ip_virtboard);
INIT_IP_VIRTBOARD(&ip_virtboard[0]);
INIT_IP_VIRTBOARD(&ip_virtboard[1]);
ip_virtboard[0].sip_msginfo_mask =
         IP_SIP_MSGINFO_ENABLE | IP_SIP_FASTSTART_CODERS_IN_OFFERED;
         /* override SIP default to enable access to message info and faststart coder info*/
ip_virtboard[1].sip_msginfo_mask =
         IP_SIP_MSGINFO_ENABLE | IP_SIP_FASTSTART_CODERS_IN_OFFERED;
         /* override SIP default to enable access to message info and faststart coder info*/
ip_virtboard[0].h323_msginfo_mask =
         IP_H323_MSGINFO_ENABLE | IP_H323_FASTSTART_CODERS_IN_OFFERED;
         /* override H.323 default to enable access to message info and faststart coder info*/
ip_virtboard[1].h323_msginfo_mask =
         IP_H323_MSGINFO_ENABLE | IP_H323_FASTSTART_CODERS_IN_OFFERED;
         /* override H.323 default to enable access to message info and faststart coder info*/
.
.
.
```

1.11.2 Accessing Fast Start Coder Information

The Global Call IP call control library includes coder information in the extra data associated with a GCEV_OFFERED event when all of the following conditions are true:

- The library was started with the fast start coder information option enabled for the appropriate protocol.
- The fast start mode is enabled.
- The call offer is a fast start offer; that is, it includes an SDP offer (SIP) or fastStart element (H.323).
- The SDP offer or fastStart element specifies at least one coder that the library supports.

When all of these conditions are true, the extra data associated with the GCEV_OFFERED event will be a GC_PARM_BLK that contains one or more parameter elements of the following type:

IPSET_CALLINFO
IPPARM_OFFERED_FASTSTART_CODER
- value = IP_CAPABILITY data structure

Each such parameter element reflects a coder specification that was contained in the call offer. If the offer contains multiple coder specifications, the order of the parameter elements in the parameter block reflects the order of the specifications in the offer message. This order reflects the remote endpoint's coder preference, with the first specification being the most preferred and the last specification being the least preferred. If any coder properties were left unspecified by the remote end, the matching fields in the corresponding IP_CAPABILITY structure will be filled in with the value GCCAP_dontCare.

If any of the four conditions described above are not true, there will be no IPSET_CALLINFO / IPPARM_OFFERED_FASTSTART_CODER parameter element in the parameter block associated with the GCEV_OFFERED.

When the IP_CAPABILITY data structure is used to convey fast start coder information, the direction field of the structure uses the following special value defines:

IP_CAP_DIR_RMTRECEIVE
: Remote coder was specified to be Receive-only.

IP_CAP_DIR_RMTTRANSMIT
: Remote coder was specified to be Transmit-only.

IP_CAP_DIR_RMTTXRX
: Remote coder was specified to be capable of both Transmit and Receive.

1.11.3 Documentation

The online bookshelf provided with System Release 6.0 CompactPCI Feature Pack 1 for Windows contains information about all system release features including features for application development, configuration, administration, and diagnostics.

For more information about the Global Call API, see the following documents:

- *Global Call API Programming Guide*
- *Global Call API Library Reference*

For features specific to IP technology, see:

- *Global Call IP Technology Guide*

Note: The online bookshelf has not been updated for this feature, so the *Global Call IP Technology Guide* does not currently contain information about retrieving coder information from fast start call offers.

1.12 Extended H.323 Nonstandard Data

The Global Call IP call control library has been updated to support data longer than 255 bytes in certain Global Call parameter elements, specifically those elements that are used for H.323 Nonstandard Data.

The maximum length of the data that can be conveyed in a Global Call parameter element is defined by a new field that has been added to the IPCCLIB_START_DATA data structure. The application is able to set the max_parm_data_size field to any value from 255 to 4096 before starting the library with **gc_Start( )**; the default value is 255 for backwards compatibility.

Applications are able to send H.323 Nonstandard Data in a number of different contexts, including:

- in a Setup message when initiating a call
- as Nonstandard Control information in a Setup message when initiating a call
- in a H.245 User Input Indication (UII) message
- in a Q.931 Facility message
- in a Registration (RAS) message
- in an H.323 Annex M Tunneled Signaling Message (as documented elsewhere in this Release Update)

The *Global Call IP Technology Guide* documents each of the situations where Nonstandard Data can be used in a separate section, and the Documentation Updates section of this Release Update contains the updated information for each of those sections.

To insert and retrieve parameter elements that may contain extended-length data, three new utility functions have been added to the Global Call library. An additional utility function has been added for copying Global Call parameter blocks, because parameter blocks that contain extended-length parameter elements do not occupy a contiguous block of memory. All four of these new utility functions are documented in the Documentation Updates section of this Release Update.

Note that the new utility functions for Global Call parameters are backwards compatible and may be used with standard parameter elements as well as parameter elements that support extended-length data. The use of the new utility functions is **mandatory**, however, whenever a GC_PARM_BLK might contain parameter elements with extended-length data.

1.13 ISDN Network Side Conformance to Network Protocol Standards ITU-T Q921 and Q931

Previously, network-side firmware was provided for back-to-back testing purposes only and was not fully qualified for operation in a deployment environment. This feature provides conformance to network protocol standards ITU-T Q921 and Q931, allowing the Intel NetStructure® DMV and DMV-A series of boards, as well as the Intel NetStructure®

DMN160TEC and DMT160TEC digital telephony interface boards, to be deployed in network-side mode. Both T1 and E1 and all media loads are supported.

Note: The symmetrical command response protocol (SYMMETRICAL_LINK), which allows user-to-user or network-to-network protocol configurations, is still supported for back-to-back testing. However, you must disable symmetrical data link operations before enabling network-side mode. Refer to the *Intel NetStructure for DM3 Architecture for CompactPCI on Windows Configuration Guide* for details.

Following are the details about the standards involved:

- Network side T1 has been tested against protocol conformance suites compliant with the ITU-T Q921 and Q931 standards and Telcordia TR-NWT-001268 specifications. The Abstract Test Suite used is based on the Telcordia TR-NWT-001268 specifications.
- Network side E1 has been tested against protocol conformance suites compliant with ITU-T Q921 and Q931 standards. The Abstract Test Suite used is compliant with ETSI standard ETS 300 402-7 and ETS 300 403-7.

The procedure for setting the parameter to network side has not changed. The **CCS_PROTOCOL_MODE** parameter sets the network user-side protocol. User-side protocol is also known as TE (terminal emulation) protocol and network-side protocol is also known as NT (network termination) protocol. Since the parameter is set by default to user side, you must change the **CCS_PROTOCOL_MODE** parameter value from 0 (User) to 1 (Network) in the .config file as described in the *Intel NetStructure for DM3 Architecture for CompactPCI on Windows Configuration Guide*.

1.14 Support for QSIG NCAS on DM3 Boards

With the Service Update, the ability to initiate Non-Call Associated Signaling (NCAS) calls is supported for the QSIG protocol (E1 or T1) on DM3 boards. The DM3 boards that support this feature are:

- Intel NetStructure® DM/V960A-4T1-cPCI
- Intel NetStructure® DM/V1200A-4E1-cPCI

The feature is only supported on media loads that use the QSIG T1 or E1 protocol, for example, ml2_qs2_qsige1.

NCAS can establish a virtual call within the network without actually associating the B channel with the call. The call only exists on the D channel, which is normally used for signaling. Once this virtual connection has been established, the customer premise equipment (CPE) can send Facility messages to the switch or terminal equipment (TE) to convey additional information. For example, Message Waiting Indicator (MWI) supplementary service information can be encoded in a Facility IE and sent in a Q.931 Setup message. (The application is responsible for encoding/decoding the Facility IE.)

Only making (outbound) NCAS calls is supported on DM3 boards. Receiving (inbound) NCAS calls (i.e., receiving the GCEV_OFFERED event) is not supported.

1.14.1 Feature Description

NCAS allows users to communicate by user-to-user signaling without setting up a circuit-switched connection. This signaling does not occupy B channel bandwidth. A temporary signaling connection is established (and cleared) in a manner similar to the control of a circuit-switched connection.

Applications must use a specific channel for NCAS calls. For E1 interfaces, this is channel 30, that is, dtiBxT30. For T1 interfaces, this is channel 23, that is, dtiBxT23. When the call is set up with the Bearer Capabilities IE indicating that it is an NCAS call, the call is sent out on the D channel, without an associated B channel. Once the NCAS connection is established, the application can transmit user-to-user messages using the call reference number (CRN) associated with the NCAS call.

With DM3 boards, the Intel Dialogic software and firmware support 8 NCAS calls per span, that is, 32 simultaneous NCAS calls per DM/V960A-4T1-cPCI or DM/V1200A-4E1-cPCI board. The 8 NCAS calls per span are **in addition** to the normal calls that you can have. For example, with T1, you can have 23 calls per span (including one on dtiBxT23), **plus** 8 NCAS calls on dtiBxT23 at the same time.

Note the following differences between NCAS implementation on Springware boards (which are not supported in System Release 6.0 CompactPCI Feature Pack 1 for Windows) and on DM3 boards:

	DM3 Boards	Springware Boards
Channel used to make NCAS calls for T1 spans	23 (dtiBxT23)	24 (dtiBxT24)
Channel used to make NCAS calls for E1 spans	30 (dtiBxT30)	30 (dtiBxT30)
Number of simultaneous NCAS calls per D channel	8 (32 total for board with 4 spans)	16

For more information about using NCAS, see the *Global Call ISDN Technology Guide*.

New Parameters

Two new parameter IDs have been added to the GCIS_SET_BEARERCHNL parameter set for setting up NCAS calls on DM3 boards:

GCIS_PARM_CODINGSTANDARD
Set to ISDN_CODINGSTD_INTL or ISDN_CODINGSTD_CCITT.

GCIS_PARM_TRANSFERCAP
Set to BEAR_CAP_UNREST_DIG.

The parameter IDs that already exist (described in the *Global Call ISDN Technology Guide*) are:

GCIS_PARM_TRANSFERMODE
Set to ISDN_ITM_CIRCUIT.

GCIS_PARM_TRANSFERRATE
Set to PACKET_TRANSFER_MODE, which is a new value for GCIS_PARM_TRANSFERRATE that has been defined for this feature.

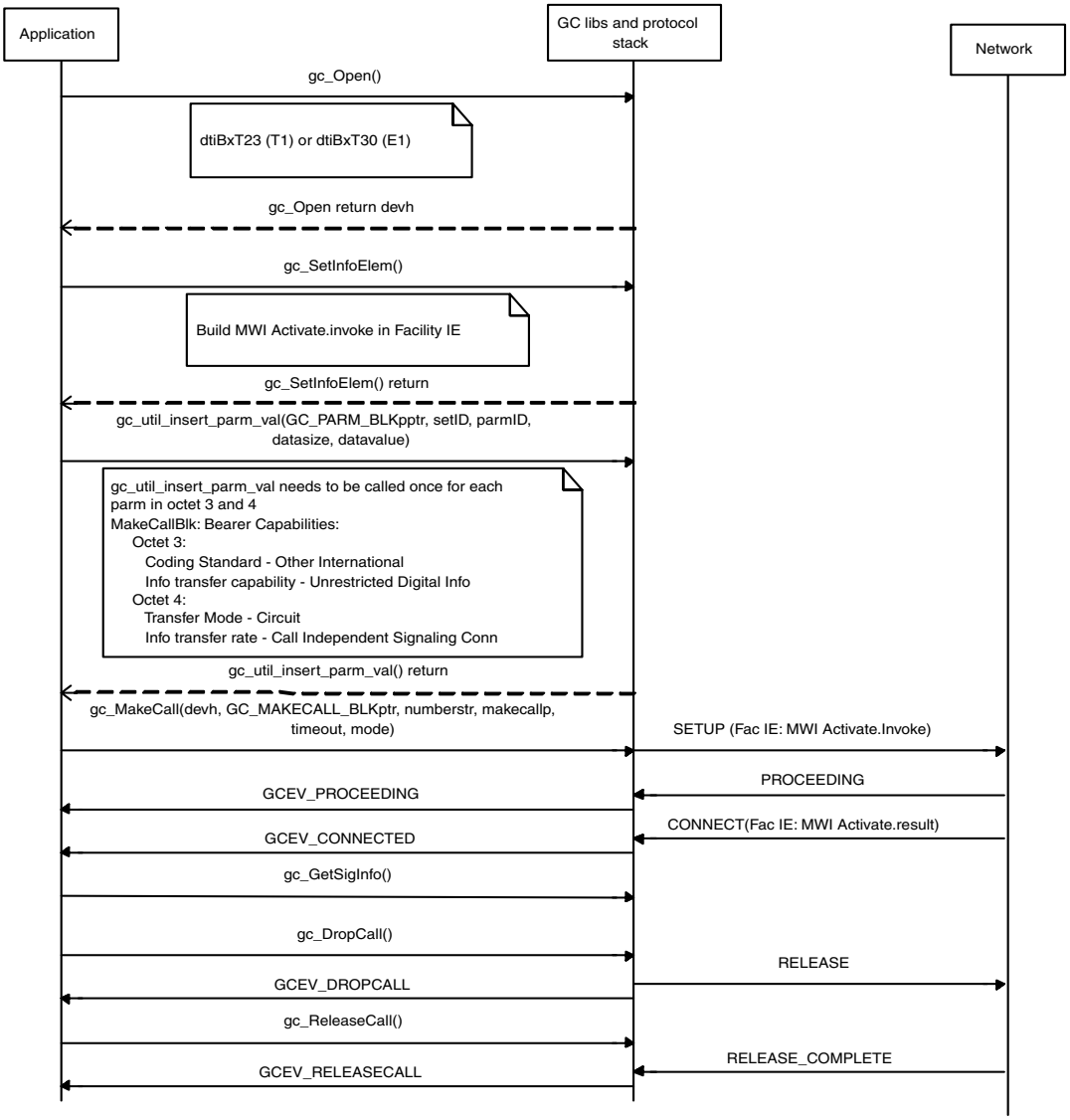
After opening the channel (T23 or T30), the **gc_util_insert_parm_val( )** function must be called to set up these four parameters. For example:

```
gc_util_insert_parm_val(pParmBlk, GCIS_SET_BEARERCHNL, GCIS_PARM_CODINGSTANDARD, sizeof(int),
ISDN_CODINGSTD_INTL);

gc_util_insert_parm_val(pParmBlk, GCIS_SET_BEARERCHNL, GCIS_PARM_TRANSFERCAP, sizeof(int),
BEAR_CAP_UNREST_DIG);

gc_util_insert_parm_val(pParmBlk, GCIS_SET_BEARERCHNL, GCIS_PARM_TRANSFERMODE, sizeof(int),
ISDN_ITM_CIRCUIT);

gc_util_insert_parm_val(pParmBlk, GCIS_SET_BEARERCHNL, GCIS_PARM_TRANSFERRATE, sizeof(int),
PACKET_TRANSFER_MODE);
```

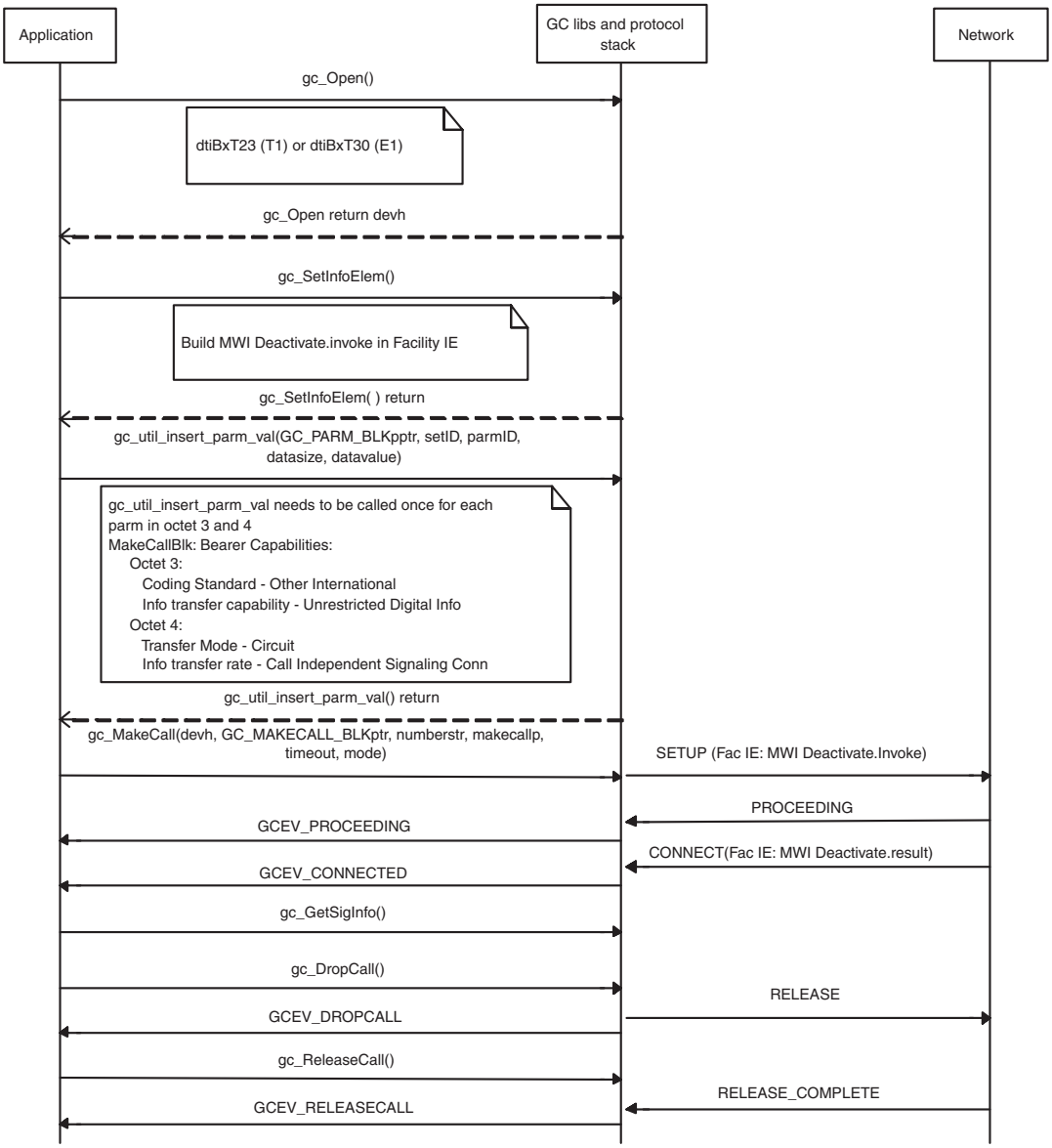
The application must also build the Facility IE (e.g., with MWI information) using **gc_SetUserInfo( )** before making the call using **gc_MakeCall( )**.

The following diagrams illustrate the API sequence for a typical MWI activation and deactivation.

Successful MWI Activate.Invoke (with Connect)



Successful MWI Deactivate.Invoke (with Connect)



1.14.2 Documentation

The online bookshelf provided with System Release 6.0 CompactPCI Feature Pack 1 for Windows contains information about all system release features including features for application development, configuration, administration, and diagnostics.

For more information about the Global Call API, see the following documents:

- *Global Call API Programming Guide*
- *Global Call API Library Reference*

For features specific to ISDN technology, see:

- *Global Call ISDN Technology Guide*

Note: The online bookshelf has not been updated for this feature, so the *Global Call ISDN Technology Guide* does not currently indicate that NCAS is supported for QSIG on DM3 boards.

1.15 New Features in Global Call Protocols Package

A number of new features have been added to the Global Call Protocols Package, which is now part of the System Release software.

The following new protocols are supported:

- Bulgaria R2
- Croatia R2
- Kuwait R2
- Lithuania R2
- Uzbekistan R2
- Korea T1/R2
- Lebanon R2
- Poland R2
- Samsung PBX Lineside E1

There are also enhancements to existing protocols:

New parameters for Nortel Meridian Lineside E1 protocol
: New parameters have been added to specify whether the protocol will wait for IDLE, wait for ReleaseGuard, and wait for SEIZEACK.

Send blocking pattern when channel is put OOS
: A new parameter, CDP_BlockOnLOOS, has been added to the CDP files for several protocols to send a blocking pattern when a channel is put out-of-service. The protocols with this new parameter are:
  - Alcatel 4400 Lineside E1
  - Alcatel VPS 4x00 Lineside

- Ericsson MD110 PBX Lineside E1
- Korea GDS Lineside E1
- Lucent Lineside E1
- NEC Lineside E1
- Nortel Meridian Lineside E1
- T1 FXS Ground Start
- United States T1 FXS/LS

Call transfer functionality

The ability to transfer calls on switches using MELCAS Lineside protocol is now supported.

An updated version of the *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* providing more detailed information about these new features will be added to the documentation bookshelf.

1.16 New Operating System Support

In addition to the supported operating systems listed in the Release Guide, the following operating system versions are now supported with this Service Update:

- Windows Server 2003 Service Pack 1 (SP1)
- Windows Server 2003 R2
- Windows 2000 Update Rollup 1 for SP4

Note: Terminal Services Application Server Mode and Active Directory Application Server Mode are not supported on any operating systems.

1.17 Enhancement to FEATURE_TABLE Data Structure for FSK Support

In the Voice Library FEATURE_TABLE data structure, ft_misc field, the FT_CALLERID bit is now being used to indicate FSK support on DM3 boards.

The FEATURE_TABLE data structure provides information about the features supported on a device. This structure is used by the **dx_getfeaturelist( )** function. On return from the function, the FEATURE_TABLE structure contains the relevant information for the device.

For further information about the FEATURE_TABLE data structure and the **dx_getfeaturelist( )** function, see the *Voice API Library Reference*. For further information about FSK, see the “Send and Receive FSK Data” chapter in the *Voice API Programming Guide*.

1.18 RH, RSS, and PHS Support on Additional Compute Platforms

With the Service Update, Redundant Host (RH) and Redundant System Slot (RSS) are now supported on the following chassis/Single Board Computer (SBC):

- Intel NetStructure® ZT5084 with ZT5550C SBC

This is in addition to the following chassis/SBCs that supported Redundant Host prior to the Service Updates:

- MPCHC5085/ZT5524A-1A (Dual CPU) chassis and single board computer
- MPCHC5085/ZT5524A-1B (Single CPU) chassis and single board computer

To use the RH and RSS capabilities on the newly supported compute platform, install the latest version of the Redundant Host Software when you install the Service Update. The Redundant Host Software is a separate setup package on the system release navigation screen. (RSS support is part of the Redundant Host Software package.)

With the Service Update, Peripheral Hot Swap (PHS) is now supported on the following chassis/Single Board Computers (SBC):

- Intel NetStructure® ZT5084 with ZT5550C SBC
- Intel NetStructure® ZT5087 with ZT5503 SBC
- Intel NetStructure® MPCHC5091 with ZT5524A-1A SBC
- Intel NetStructure® MPCHC5091 with ZT5524A-1B SBC

These are in addition to the following chassis/SBCs that supported PHS prior to the Service Updates:

- Intel NetStructure® MPCHC5085/ZT5524A-1A
- Intel NetStructure® MPCHC5085/ZT5524A-1B
- Advantech* MIC3038/MIC3358
- Motorola* CPX8216/CPV5350v
- Westek* P5100/ Advantech MIC-3358

To use PHS capabilities with the newly supported compute platforms, install the latest version of the Hot Swap Kit software when you install the Service Update. The Hot Swap Kit software is a separate setup package on the system release navigation screen.

For further information about installing the Hot Swap Kit and Redundant Host Software, see the Software Installation Guide on the online bookshelf.

Note: RH, RSS, and PHS are supported on Windows 2000 SP4 only.

1.19 Mixing ISDN and Clear Channel on a DMN160TEC or DMT160TEC Board

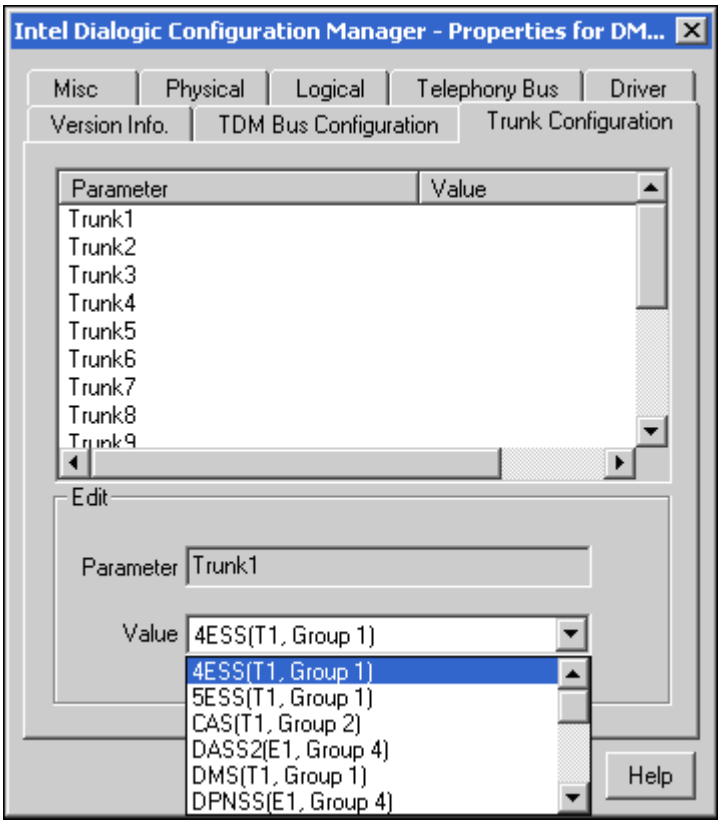
With the Service Update, you can now mix ISDN trunks and clear channel trunks on the same Intel NetStructure® DMN160TEC or DMT160TEC digital telephony interface board using the Trunk Configuration property sheet of the configuration manager (DCM). Previously, you had to edit the CONFIG file to accomplish this.

1.19.1 Feature Description

The DCM Trunk Configuration property sheet has been modified to enable you to mix ISDN and clear channel trunks without having to edit the CONFIG file. Clear channel values have been added and all values have been assigned to Groups. The clear channel values are E1CC, ISDNE1CC, ISDNT1CC, and T1CC. The Group number ensures that you only use compatible values on a given board because you cannot complete the configuration process successfully unless all the values you select for a given board are from the same group. Following are updated guidelines on using the Trunk Configuration property sheet.

Trunk Configuration Property Sheet

The Trunk Configuration property sheet contains parameters for configuring the interfaces on DMN160TEC and DMT160TEC boards. Each parameter is in the format Trunk*x*, where *x* denotes the trunk number (1 to 16).



Note: The Trunk Configuration property sheet only applies to DMN160TEC and DMT160TEC boards.

The **Trunk1** (to **Trunk16**) parameter specifies a protocol or clear channel value and a line type to use for each set of four interfaces on a DMN160TEC or DMT160TEC board. The values you choose for a given board must all have the same Group number.

Values:

- 4ESS(T1, Group 1) [default]
- 5ESS(T1, Group 1)
- CAS(T1, Group 2) (Supported on DMT160TEC board only; not on DMN160TEC board)
- DASS2(E1, Group 4)
- DMS(T1, Group 1)
- DPNSS(E1, Group 4)

- E1CC(E1, Group 3)
- ISDNE1CC(E1, Group 1)
- ISDNT1CC(T1, Group 1)
- NET5(E1, Group 1)
- NI2(T1, Group 1)
- NTT(T1, Group 1)
- QSIGE1(E1, Group 1)
- QSIGT1(T1, Group 1)
- R2MF(E1, Group 3)
- T1CC(T1, Group 2)

Guidelines: Each set of four trunks (1-4, 5-8, 9-12, and 13-16) must be assigned one of the values listed above. You can assign a different value to each set of four trunks, but all the values you use must have the same Group number. For example, if you choose the 4ESS protocol (which is in Group 1) for the first set of four trunks, you can only assign protocols or clear channel values in Group 1 (such as ISDNE1CC) to the other sets of trunks. You can mix T1 and E1 line types as long as they are in the same Group.

You can mix ISDN trunks and clear channel trunks on the same DMN160TEC or DMT160TEC board as long as you follow the above guidelines. The following values are for clear channel signaling: E1CC, ISDNE1CC, ISDNT1CC, and T1CC.

The following table shows the protocols in each group and also indicates their line type.

Group 1		Group 2	Group 3	Group 4
T1	E1	T1	E1	E1
4ESS	ISDNE1CC	CAS*	E1CC	DASS2
5ESS	NET5	T1CC	R2MF	DPNSS
DMS	QSIGE1			
ISDNT1CC				
NI2				
NTT				
QSIGT1				

*CAS is supported on DMT160TEC board only; not on DMN160TEC board.

Note: After changing a trunk's configuration, the board must be re-initialized (started) for the configuration to take effect.

1.19.2 Documentation

The online bookshelf provided with System Release 6.0 CompactPCI Feature Pack 1 for Windows contains information about all system release features including features for application development, configuration, administration, and diagnostics.

For more information about configuring DMN160TEC and DMT160TEC boards, see the following document:

- *Intel NetStructure for DM3 Architecture for CompactPCI on Windows Configuration Guide*

Note: The online bookshelf has not been updated for this feature. The information in the Configuration Guide about editing the CONFIG file to configure trunks for clear channel signaling is superseded by the information in this Release Update.

1.20 Using H.323 Annex M Tunneled Signaling Messages

With the Service Update, the Global Call IP call control library supports the tunneled signaling message capability that is documented in Annex M of the ITU-T recommendations for H.323. This capability allows DSS/QSIG/ISUP messages to be encapsulated in common H.225 call signaling messages. Note that this tunneled message capability is separate and distinct from H.245 tunnelling.

The tunneled signaling message capabilities are described in the following topics:

- Tunneled Signaling Message Overview
- Sending Tunneled Signaling Messages
- Enabling Reception of Tunneled Signaling Messages
- Receiving Tunneled Signaling Messages
- Parameter Reference
- Data Structure Reference
- Documentation

1.20.1 Tunneled Signaling Message Overview

The ITU-T Annex M recommendation specifies that tunneled signaling message fields may be contained in a number of different H.225 messages, including SETUP, INFORMATION, CALL PROCEEDING, ALERTING, PROGRESS, NOTIFY, CONNECT, RELEASE COMPLETE, and FACILITY.

The Global Call implementation of tunneled signaling messages allows applications to send tunneled messages only in H.225 SETUP messages, as sent by the **gc_MakeCall( )** function. Only one tunneled signaling message can be sent per SETUP message.

The reception of tunneled signaling messages via Global Call is an optional feature that can only be enabled when starting the virtual board. When the feature is enabled, tunneled message fields can be retrieved from any of the H.225 messages specified in Annex M. An application has no ability to specify which message types it wishes to receive tunneled signaling message in; if there is any possibility that the remote agent could be a non-Global Call application, the local application must be prepared to handle tunneled messages in any of the specified H.225 message types.

Tunneled signaling messages are constructed by configuring a GC_PARM_BLK with parameter elements that contain protocol identification, message content, and nonstandard data fields. The protocol identification can use either a protocol object ID or an alternate identification data structure, IP_TUNNELPROTOCOL_ALTID. As in other Global Call implementations of nonstandard data, the H.221 protocol can be specified, or the nonstandard data can be identified via an object ID. When the GC_PARM_BLK is configured, it is passed to the **gc_MakeCall( )** function as part of the GC_MAKECALL_BLK data structure, at which point the library and H.323 stack package the supplied data as tunneled signaling message fields in the H.225 SETUP message sent by the function call.

Note: The **gc_SetUserInfo( )** and **gc_SetConfigData( )** functions **cannot** be used to configure the tunneled signaling message parameters, and the **gc_Extension( )** function cannot be used to send a message that contains tunneled signaling message fields. The configured parameter data must be passed directly to **gc_MakeCall( )**.

When reception of tunneled signaling messages is enabled as described in Section 1.20.3, “Enabling Reception of Tunneled Signaling Messages”, on page 58, applications must register to receive the messages using the **gc_Extension( )** function. When any H.225 message containing a tunneled signaling message is received, the library generates an asynchronous GCEV_EXTENSIONCMPLT completion event, which includes the tunneled signaling message information in the metaevent data. Tunneled signaling messages can only be retrieved within a call (the application must use a valid CRN when registering to receive tunneled signaling messages), but the call can be in any state.

1.20.2 Sending Tunneled Signaling Messages

The process of sending a tunneled signaling message begins by composing a GC_PARM_BLK that contains parameter elements for the message protocol, the message content, and any nonstandard data.

The first parameter element identifies the message protocol. It must be **one** of the following two forms:

IPSET_TUNNELEDSIGNALMSG
IPPARM_TUNNELEDSIGNALMSG_PROTOCOL_OBJID

- value = protocol object ID string

IPSET_TUNNELEDSIGNALMSG
IPPARM_TUNNELEDSIGNALMSG_ALTERNATEID

- value = alternate protocol ID information in an IP_TUNNELPROTOCOL_ALTID data structure

The second parameter element contains the actual message content:

IPSET_TUNNELEDSIGNALMSG
IPPARM_TUNNELEDSIGNALMSG_CONTENT

- value = actual message content

If the tunneled signal message includes nonstandard data, the GC_PARM_BLOCK needs to contain two additional parameter elements. These parameters should not be inserted in the GC_PARM_BLK if nonstandard data is not being sent in the message. The first parameter element for nonstandard data is:

IPSET_TUNNELEDSIGNALMSG
IPPARM_TUNNELEDSIGNALMSG_NSDATA_DATA

- value = actual nonstandard data, max. length = max_parm_data_size (configured at library start-up)

The second parameter element for nonstandard data uses **one** of the following two forms:

IPSET_TUNNELEDSIGNALMSG
IPPARM_TUNNELEDSIGNALMSG_NSDATA_OBJID

- value = nonstandard data object ID string

IPSET_TUNNELEDSIGNALMSG
IPPARM_TUNNELEDSIGNALMSG_NSDATA_H221NS

- value = H.221 nonstandard data information in an IP_H221NONSTANDARD data structure

Note: In practice, applications may not be able to utilize full maximum parameter length configured in max_parm_data_size for nonstandard data content. The H.323 stack limits the overall size of messages to be max_parm_data_size + 512 bytes, which must contain the tunneled signaling message content as well as the nonstandard data.

Once the GC_PARM_BLK is composed, the block is included in a GC_MAKECALL_BLK, and that block is then passed as a parameter in a call to **gc_MakeCall( )**.

The following code example illustrates the process of composing the parameter block for a tunneled signaling message.

```
#include <stdio.h>
#include <string.h>
#include <gcip.h>
#include <.h>

void main()
{
   IP_TUNNELPROTOCOL_ALTID tsmTpAltId;
   IP_H221NONSTANDARD      tsmH221NS;
   GC_PARM_BLKP            pParmBlock;

   /*. . Main Processing….*/

   char *pTP_Oid = "11 22 33 44 66";
                  // Note that the Object Id strings must must be in the correct ASN.1 format.
   char *pMsgContent = "00 11 22 33 44 55";

   char TP_AltID_Type[]    = "Tunneled Protocol Alternate ID protocol type";
   char TP_AltID_Variant[] = "Tunneled Protocol Alternate ID protocol variant";
   char TP_AltID_SubId[]   = "Tunneled Protocol Alternate ID subidentifier - User";

   char *ptsmNSData_Data   = "Tunneled Signaling Message Non Standard Data";
   char *ptsmNSData_Oid    = "99 88 77 11 03";
                // Note that the Object Id strings must be in the correct ASN.1 format
                // otherwise it may cause problems in the RV Stack.
```

```
/* Initialize the structures before use */
INIT_IP_TUNNELPROTOCOL_ALTID (&tsmTpAltId);

strcpy(tsmTpAltId.protocolType, TP_AltID_Type);
tsmTpAltId.protocolTypeLength = strlen(TP_AltID_Type);
strcpy(tsmTpAltId.protocolVariant, TP_AltID_Variant);
tsmTpAltId.protocolVariantLength = strlen(TP_AltID_Variant);
strcpy(tsmTpAltId.subIdentifier, TP_AltID_SubId);
tsmTpAltId.subIdentifierLength = strlen(TP_AltID_SubId);

tsmH221NS.country_code = 91;
tsmH221NS.extension = 202;
tsmH221NS.manufacturer_code = 11;

choiceOfTSMProtocol = 1;
    /* App decides whether to use the tunneled signaling message Protocol Object ID/ AltID */
choiceOfNSData = 1;
    /* App decides which type of object identifier to use for TSM NS Data */

/* setting tunneled signaling message fields */

if (choiceOfTSMProtocol)
/* App decides the choice of the tunneled signaling msg protocol object identifier */
/* It cannot set both objid & alternate id */
{
   gc_util_insert_parm_ref(&pParmBlock,
                           IPSET_ TUNNELEDSIGNALMSG,
                           IPPARM_TUNNELEDSIGNALMSG_PROTOCOL_OBJID,
                           (unsigned char) (strlen(pTP_Oid) + 1),
                           pTP_Oid);
}

else
{
   gc_util_insert_parm_ref(&pParmBlock,
                           IPSET_ TUNNELEDSIGNALMSG,
                           IPPARM_TUNNELEDSIGNALMSG_ALTERNATEID,
                           (unsigned char)sizeof(IP_TUNNELPROTOCOL_ALTID),
                           & tsmTpAltId);
}

gc_util_insert_parm_ref(&pParmBlock,
                        IPSET_TUNNELEDSIGNALMSG,
                        IPPARM_TUNNELEDSIGNALMSG_CONTENT,
                        (unsigned char)(strlen(pMsgContent)+1),
                        pMsgContent);

/* Now fill in the Tunneled Signaling message Non Standard data fields */
/* Note the use of the extended gc_util function because NSD data may exceed 255 bytes */
gc_util_insert_parm_ref_ex(&pParmBlock,
                           IPSET_TUNNELEDSIGNALMSG,
                           IPPARM_TUNNELEDSIGNALMSG_NSDATA_DATA,
                           (unsigned long)(strlen(ptsmNSData_Data)+1),
                           ptsmNSData_Data);

if (choiceOfNSData)
/* App decides the CHOICE of Non Standard OBJECTIDENTIFIER. */
/* It cannot set both objid & H221 */
{
   gc_util_insert_parm_ref(&pParmBlock,
                           IPSET_TUNNELEDSIGNALMSG,
                           IPPARM_ TUNNELEDSIGNALMSG_NSDATA_OBJID,
                           (unsigned char) (strlen(ptsmNSData_Oid)+1),
                           ptsmNSData_Oid);
}
```

```
else
{
   gc_util_insert_parm_ref(&pParmBlock,
                           IPSET_TUNNELEDSIGNALMSG,
                           IPPARM_ TUNNELEDSIGNALMSG_NSDATA_H221NS,
                           (unsigned char)sizeof(IP_H221NONSTANDARD),
                           & tsmH221NS);
}

/*. .. Continue Main processing. … call gc_MakeCall() */
```

1.20.3 Enabling Reception of Tunneled Signaling Messages

The ability to retrieve tunneled signaling messages from inbound H.225 messages is an optional feature that can be enabled or disabled at the time the **gc_Start( )** function is called.

The **INIT_IPCCLIB_START_DATA( )** and **INIT_IP_VIRTBOARD( )** functions, which must be called before **gc_Start( )**, populate the IPCCLIB_START_DATA and IP_VIRTBOARD structures, respectively, with default values. The default value of the h323_msginfo_mask field in the IP_VIRTBOARD structure does not enable either access to Q.931 message information elements or the ability to receive tunneled signaling messages. To enable either or both of these features for an IPT device, the default value of the h323_msginfo_mask field must be overridden with a value that represents the appropriate logical combination of the two defined mask values. To enable reception of tunneled signaling messages, the value IP_H323_ANNEXMMSG_ENABLE must be set. The following code snippet enables Q.931 message IE access on two virtual boards and enables tunneled signaling messages on the second board only:

```
INIT_IPCCLIB_START_DATA(&ipcclibstart, 2, ip_virtboard);
INIT_IP_VIRTBOARD(&ip_virtboard[0]);
INIT_IP_VIRTBOARD(&ip_virtboard[1]);
ip_virtboard[0].h323_msginfo_mask = IP_H323_MSGINFO_ENABLE;
                                    /* override Q.931 message default */
ip_virtboard[1].h323_msginfo_mask = IP_H323_MSGINFO_ENABLE | IP_H323_ANNEXMMSG_ENABLE;
                                    /* override Q.931 message and TSM defaults */
```

Note: Once the tunneled signaling message feature is enabled on a virtual board, there is no way to disable the retrieval of the messages other than reconfiguring and restarting the virtual board.

1.20.4 Receiving Tunneled Signaling Messages

Assuming that reception of tunneled signaling messages was enabled when the virtual board was started, the application registers to receive within each call using the **gc_Extension( )** function and the extension ID IPEXTID_GETINFO.

The parameters for the **gc_Extension( )** function call must be set up as follows:

- **target_type** must be GCT_GCLIB_CRN. The function cannot be called for a line device.
- **target_id** must be a valid CRN. The call can be in any state.
- **ext_id** must be IPEXTID_GETINFO

- **parmblkp** must point to a GC_PARM_BLK that contains a parameter with the set ID IPSET_TUNNELEDSIGNALMSG and IPPARM_TUNNELEDSIGNALMSG_CONTENT parameter ID. This is the only field that will always be present in every received tunneled signaling message; the library automatically ensures that all tunneled signaling message fields that actually exist in the message are retrieved as long as this one parameter is present in the GC_PARM_BLK.
- **retblkp** must be a valid pointer to a GC_PARM_BLK
- **mode** must be EV_ASYNC

The following code illustrates a typical registration process.

```
int getTSMInfo(CRN crn)
{
   GC_PARM_BLKP gcParmBlk = NULL;
   GC_PARM_BLKP retParmBlk;
   int frc;

   frc = gc_util_insert_parm_val(&gcParmBlk,
                                 IPSET_TUNNELEDSIGNALMSG,
                                 IPPARM_TUNNELEDSIGNALMSG_CONTENT,
                                 sizeof(int),1);

   if (GC_SUCCESS != frc)
   {
      return GC_ERROR;
   }

   frc = gc_Extension (GCTGT_GCLIB_CRN,
                       crn,
                       IPEXTID_GETINFO,
                       gcParmBlk,
                       &retParmBlk,
                       EV_ASYNC);
   if (GC_SUCCESS != frc)
   {
      return GC_ERROR;
   }

   gc_util_delete_parm_blk(gcParmBlk);

   return GC_SUCCESS;

}
```

After this registration, when an H.225 message containing a tunneled signaling message is received by the library, it generates an asynchronous GCEV_EXTENSIONCMPLT completion event. The extevtdatap field in the METAEVENT structure for this event is a pointer to an EXTENSIONEVTBLK structure, which in turn contains a GC_PARM_BLK that contains the fields of the received tunneled signaling message. Applications are then able to extract the data of interest using code similar to the following example.

Notes:1. The application must take care to retrieve the Annex M Message information from any incoming H.225 message before the next H.225 message arrives; if the new message also contains TSM information, that new TSM will overwrite the prior information.

2. The overall message size that the Global Call H.323 stack can handle is defined as max_parm_data_size (which is configured at library startup) + 512 bytes. Any message that is received which exceeds this length is truncated.

3. Parameter values that are contained in a GC_PARM_BLK are subject to maximum length limits that are defined for each parameter type; for example, tunneled signaling message content is limited to 255 bytes, and nonstandard data is limited to max_parm_data_size (which is configured at library startup). Any data received in a TSM that exceeds these defined limits is truncated without notification to the application.

4. The application should use the extended **gc_util_..._ex( )** functions when extracting parameters from a GC_PARM_BLK that contains TSM contents because the Global Call parameter for nonstandard data supports data length that may exceed 255 bytes.

```
int OnExtension(GC_PARM_BLKP parm_blk, CRN crn)
{
   INIT_GC_PARM_DATA_EXT(*parmp);
   retval = gc_util_next_parm_ex(parm_blk, parmp);

   if (retval == GC_ERROR)
   {
      return GC_ERROR;
   }

   while (retval != EGC_NO_MORE_PARMS)
   {
      switch (parmp->set_ID)
      {
         case IPSET_TUNNELEDSIGNALMSG:
            switch (parmp->parm_ID)
            {
               case IPPARM_TUNNELEDSIGNALMSG_CONTENT:
                  printf("\tReceived extension data (TSM) Msg Content: %s\n",
                         parmp->value_buf);
                  break;

               case IPPARM_TUNNELEDSIGNALMSG_PROTOCOL_OBJID:
                  printf("\tReceived extension data (TSM) PROTOCOL_OBJID: %s\n",
                         parmp->value_buf);
                  break;

               case IPPARM_TUNNELEDSIGNALMSG_ALTERNATEID:
               {
                  if(parmp->value_size == sizeof(IP_TUNNELPROTOCOL_ALTID))
                  {
                     IP_TUNNELPROTOCOL_ALTID *ptsmTpAltId;
                     ptsmTpAltId = (IP_TUNNELPROTOCOL_ALTID *)(&(parmp->value_buf));
                     printf("\tReceived extension data (TSM) Protocol Alt id:
                            Type=%s, Variant=%s, Sub Id=%s\n",
                            ptsmTpAltId->protocolType,
                            ptsmTpAltId->protocolVariant,
                            ptsmTpAltId->subIdentifier);
                  }
               }
               break;
```

```
                case IPPARM_TUNNELEDSIGNALMSG_NSDATA_DATA:
                   printf("\tReceived extension data (TSM NSDATA) DATA: %s\n",
                          parmp->value_buf);
                   break;

                case IPPARM_TUNNELEDSIGNALMSG_NSDATA_OBJID:
                   printf("\tReceived extension data (TSM NSDATA) OBJID: %s\n",
                          parmp->value_buf);
                   break;

                case IPPARM_TUNNELEDSIGNALMSG_NSDATA_H221NS:
                {
                   if(parmp->value_size == sizeof(IP_H221NONSTANDARD))
                   {
                      IP_H221NONSTANDARD *pH221NonStandard;
                      pH221NonStandard = (IP_H221NONSTANDARD *)(&(parmp->value_buf));
                      printf("\tReceived extension data (NSDATA) h221:CC=%d, Ext=%d, MC=%d\n",
                             pH221NonStandard->country_code,
                             pH221NonStandard->extension,
                             pH221NonStandard->manufacturer_code);
                   }
                }
                break;

          default:
             printf("\tReceived unknown (TSM NSDATA) extension parmID %d\n",
                    parmp->parm_ID);
             break;
       }
       break;

   retval = gc_util_next_parm_ex(parm_blk,parmp);
   }

}
```

1.20.5 Parameter Reference

The IPSET_TUNNELEDSIGNALMSG parameter set and parameter IDs are defined in the *gcip.h* header file. The parameter IDs are:

- IPPARM_TUNNELEDSIGNALMSG_ALTERNATEID
- IPPARM_TUNNELEDSIGNALMSG_CONTENT
- IPPARM_TUNNELEDSIGNALMSG_NSDATA_DATA
- IPPARM_TUNNELEDSIGNALMSG_NSDATA_H221NS
- IPPARM_TUNNELEDSIGNALMSG_NSDATA_OBJID
- IPPARM_TUNNELEDSIGNALMSG_PROTOCOL_OBJID

All parameters are supported for H.323 only. They are sent via the **gc_MakeCall( )** function and retrieved via the GCEV_EXTENSIONCMPLT(IPEXTID_RECEIVEMSG) event type.

Detailed information about each of the parameters in the parameter set is provided in Table 1-1.

Table 1-1. IPSET_TUNNELEDSIGNALMSG Parameter Set

Parameter IDs	Data Type & Size	Description	SIP/ H.323
IPPARM_ TUNNELEDSIGNALMSG_ ALTERNATEID	Type: IP_TUNNEL PROTOCOL_ ALTID Size: sizeof(IP_TUNNEL PROTOCOL_ ALTID)	Used to contain a tunneled protocol alternate identifier in a tunneled signaling message (TSM). Either this or the tunneled protocol object ID can exist in a TSM. If the application is using a tunneled protocol object ID when sending a TSM, this parameter should not be inserted in the GC_PARM_BLK.	H.323 only
IPPARM_ TUNNELEDSIGNALMSG_ CONTENT	Type: string † Max length: MAX_IE_ LENGTH (255)	Used to contain any data content of a tunneled signaling message (TSM), which is a sequence of octet strings.	H.323 only
IPPARM_ TUNNELEDSIGNALMSG_ NSDATA_DATA	Type: string † Max length: max_parm_data _size ‡ (configured via IPCCLIB_START _DATA)	Used to contain any nonstandard data in a tunneled signaling message (TSM). If no nonstandard data is being sent in a TSM, this parameter should not be inserted in the GC_PARM_BLK.	H.323 only
IPPARM_ TUNNELEDSIGNALMSG_ NSDATA_H221NS	Type: IP_H221 NONSTANDARD Size: sizeof(IP_H221NON STANDARD)	Used to contain an H.221 nonstandard data identifier in a tunneled signaling message (TSM). Either this ID or the nonstandard data object ID can exist in a TSM's nonstandard data. If nonstandard data is not being sent, or if a nonstandard data object ID is being used when sending a TSM, this parameter should not be inserted in the GC_PARM_BLK.	H.323 only
IPPARM_ TUNNELEDSIGNALMSG_ NSDATA_OBJID	Type: string † Max length: MAX_NS_ PARAM_OBJID_ LENGTH (40)	Used to contain a nonstandard data object identifier in a tunneled signaling message (TSM). Either this ID or an H.221 nonstandard data ID can exist in a TSM's nonstandard data. If nonstandard data is not being sent, or if an H.221 nonstandard data ID is being used when sending a TSM, this parameter should not be inserted in the GC_PARM_BLK.	H.323 only
IPPARM_ TUNNELEDSIGNALMSG_ PROTOCOL_OBJID	Type: string † Max length: MAX_TSM_ POID_PARAM_ LENGTH (128)	Used to contain a tunneled protocol object identifier in a tunneled signaling message (TSM). Either this or the tunneled protocol alternate ID can exist in a TSM. If the application is using an alternate identifier when sending a TSM, this parameter should not be inserted in the GC_PARM_BLK.	H.323 only

† For parameters with data of type String, the length in a GC_PARM_BLK is the length of the data string plus 1.
‡ The full maximum length that is configured may not be usable in practice because the H.323 stack limits total message size to max_parm_data_size + 512 bytes. Longer messages are truncated without notification to the application.

1.20.6 Data Structure Reference

IP_VIRTBOARD

There is a change to the IP_VIRTBOARD data structure for using H.323 Annex M tunneled signaling messages. There is a new value, IP_H323_ANNEXMMSG_ENABLE, for the h323_msginfo_mask field. This field is now defined as follows:

h323_msginfo_mask (structure version ≥ 0x103 only)

Enables and disables reception of H.323 message information. Access is disabled by default. The following mask values, which may be OR'ed together, are defined to enable the features:

- IP_H323_ANNEXMMSG_ENABLE – enable reception of H.323 Annex M tunneled signaling messages in H.225 messages
- IP_H323_MSGINFO_ENABLE – enable access to H.323 message information fields

IP_TUNNELPROTOCOL_ALTID

Reference information about the IP_TUNNELPROTOCOL_ALTID data structure, which is used when sending tunneled signaling messages, is given below.

IP_TUNNELPROTOCOL_ALTID

```
typedef struct
{
    unsigned long  version;
    char           protocolType[MAX_TSM_ALTID_VARS_LENGTH];
    int            protocolTypeLength;
    char           protocolVariant[MAX_TSM_ALTID_VARS_LENGTH];
    int            protocolVariantLength;
    char           subIdentifier[MAX_TSM_ALTID_VARS_LENGTH];
    int            subIdentifierLength;
} IP_TUNNELPROTOCOL_ALTID;
```

Description

The IP_TUNNELPROTOCOL_ALTID data structure is used for H.323 Annex M tunneled signaling protocol alternate protocol ID information.

Applications should use the **INIT_IP_TUNNELPROTOCOL_ALTID( )** function to initialize the structure with the correct version number and initial field values.

Field Descriptions

The fields of the IP_TUNNELPROTOCOL_ALTID data structure are described as follows:

version
: The version number of the data structure. The correct value is set by the **INIT_IP_TUNNELPROTOCOL_ALTID( )** initialization function and should not be overridden.

protocolType
: A string that identifies the tunneled protocol type. Maximum length is 64.

protocolTypeLength
: The length of the protocolType string.

protocolVariant
: A string that identifies the tunneled protocol variant. Maximum length is 64.

protocolVariantLength
: The length of the protocolVariant string.

subIdentifier
: A string that provides additional tunneled protocol identification.

subIdentifierLength
: The length of the subIdentifier string.

1.20.7 Documentation

The online bookshelf provided with System Release 6.0 CompactPCI Feature Pack 1 for Windows contains information about all system release features including features for application development, configuration, administration, and diagnostics.

For more information about the Global Call API, see the following documents:

- *Global Call API Programming Guide*
- *Global Call API Library Reference*

For features specific to IP technology, see:

- *Global Call IP Technology Guide*

Note: The online bookshelf has not been updated for this feature, so the *Global Call IP Technology Guide* does not currently contain information about using H.323 Annex M tunneled signaling messages.

1.21 New Option for dm3post Utility

With the Service Update, the dm3post diagnostic utility now provides an option to run POST on a chassis level. By using the chassis option (-c), dm3post will retrieve the results of the last run POST for all DM3 boards in the chassis. By using the chassis option with the reset (-r) option, you can run POST on all DM3 boards in the system.

When using the chassis option, it is not necessary to provide the bus and slot numbers. Any option other than the reset option will be ignored when using the chassis option. In addition to output on the screen, more detailed output is logged to a log file, dm3post.log, by default.

For more information about the dm3post utility, see the Diagnostics Guide.

1.22 Tracing CAS Signaling

With the Service Update, inbound and outbound R2MF tones and CAS bit transitions/states can now be traced using existing Global Call APIs and a new event. This allows developers to determine the root cause of protocol issues in a system that uses Intel NetStructure® DMT160TEC digital telephony interface boards.

For detailed information about tracing CAS signaling using Global Call, see the Diagnostics Guide.

1.23 ISDN Trace Capability on Multiple Trunks

With the Service Update, the capture of ISDN D-channel trace information can now be dynamically started and stopped via Global Call APIs, and logs can be collected on two or more trunks at the same time. Previously, the only available tool for collecting ISDN trace information (isdntrace) could not be run on more than one trunk. This trace information allows developers to determine the root cause of protocol issues in a system that uses Intel NetStructure® DMT160TEC or DMN160TEC digital telephony interface boards (no other boards support this feature).

For detailed information about tracing multiple ISDN trunks using Global Call, see the Diagnostics Guide.

Note: Enabling ISDN tracing on a higher number of trunks will cause the call performance to be severely degraded and must not be left permanently enabled in a production environment.

1.24 Support for DCB Conferencing Library in RTF Tool

With the Service Update, the Runtime Trace Facility (RTF) diagnostics tool now supports tracing of the audio conferencing (DCB) API library.

For information about using the RTF tool, see the Diagnostics Guide.

1.25 Enhanced Volume Control for Conferencing

The Service Update provides the following enhancements to the audio conferencing (DCB) API library for DM/V-A Series boards:

- Ability to adjust the output or speaker volume of a conference on an individual conferee basis by setting specific dB levels using a new API function. The output or speaker volume refers to the volume that a conferee hears from the conference. The conference moderator can raise, lower, or mute the output volume level for any conferee as needed. Previously, volume level could only be adjusted on a board basis using DTMF digits.
- Ability to adjust default automatic gain control (AGC) parameters in the configuration file as a way to control input volume. Input volume refers to the volume of a specific conferee's input to a conference. Although default AGC parameters are set at optimal levels, you can modify the default values on a board basis during board initialization as needed. Previously, AGC values could not be controlled.

Note: These enhancements are available on all media loads, including the basic conferencing media load (ML9C).

1.25.1 Feature Description

The new functions and parameters for volume control are described in the following sections:

- New Conferencing API Functions
- New Configurable AGC Parameters for Conferencing

1.25.1.1 New Conferencing API Functions

Two new functions are provided to adjust and retrieve the output volume at a dB level for any conferee (conference party):

- **dcb_SetPartyParm( )**
- **dcb_GetPartyParm( )**

For information about these functions, see the *Dialogic® Audio Conferencing API Library Reference*.

1.25.1.2 New Configurable AGC Parameters for Conferencing

As an alternate way to control input volume of a conference, you can change the default values of the automatic gain control (AGC) parameters in the CONFIG file. To specify new values, you must manually enter these parameters in the CONFIG file in the [0x3b] section. The new values that you specify take effect on a board basis. The parameters are:

- CSUMS_AGC_k_Def (AGC K Constant)
- CSUMS_AGC_low_threshold (AGC Noise Level Lower Threshold)
- CSUMS_AGC_max_gain (AGC Maximum Gain)

The new AGC parameters are described in more detail next.

Note: For background information helpful in understanding AGC, see Figure 6, AGC Gain vs. Input Average, in the *Intel NetStructure for DM3 Architecture for CompactPCI on Windows Configuration Guide*.

CSUMS_AGC_k_Def (AGC K Constant)

Number: 0x3B13

Description: The **CSUMS_AGC_k_Def** parameter is the target output level to the TDM bus. Note that K is the average level for the output.

CSUMS_AGC_k_Def is defined as: K * 2^{23}. K is defined as $10^{(\text{output level in dB})/20}$. Multiplying by 2^{23} converts the value into a linear 24-bit value that accommodates the 24-bit DSPs used on DM3 boards. Therefore, K = 0.0562341 corresponds to -25 dB average, since 0.0562341= $10^{(-25/20)}$. Note that -25 dB average would result in -18 dBm level of the analog output signal.

Values: 0x040C37 to 0x1ABE34 (-30.0 dB to -13.6 dB output levels)

Default Value: 0x0732AE (-25dB)

Guidelines: It is recommended that the value be set in the range of -30.0 dB to -13.6 dB. Higher values may result in strong peak to average compression if it is enabled or just severe clipping if peak to average compression is disabled. The higher output level may also result in higher echo.

Here is a sample calculation to obtain a hexadecimal value of **CSUMS_AGC_k_Def** for an output level of -13.6 dB:

$(10^{(-13.6/20)})* 2^{23} = 0x1ABE34$

CSUMS_AGC_low_threshold (AGC Noise Level Lower Threshold)

Number: 0x3B1F

Description: The **CSUMS_AGC_low_threshold** parameter defines the upper threshold for noise level estimates. Any signal above this threshold will be considered speech. Thus, this threshold should be set quite high in order to let the AGC algorithm determine when there are voiced and unvoiced periods. The parameter is given in terms of the average level.

CSUMS_AGC_low_threshold is defined as: $10^{(\text{output level in dB})/20} * 2^{23}$. Multiplying by 2^{23} converts the value into a linear 24-bit value that accommodates the 24-bit DSPs used on DM3 boards.

Values: 0x0020C5 to 0x0732AE (-60 dB to -25 dB)

Default Value: 0x0147AE (-40 dB)

Guidelines: It is recommended that the value be set in the range of -60 dB to -40 dB. Do not exceed the AGC high threshold which is set to -34.6 dB in the current DM3 system.

Here is a sample calculation to get a hexadecimal value of **CSUMS_AGC_low_threshold** for a noise threshold level of -50 dB_{avg}:

$10^{(-50/20)} * 2^{23} = 0x00679F$

CSUMS_AGC_max_gain (AGC Maximum Gain)

Number: 0x3B1E

Description: The **CSUMS_AGC_max_gain** parameter defines the maximum gain divided by 32. This parameter controls the maximum possible gain applied by the AGC algorithm. The ratio, **CSUMS_AGC_k_Def** over **CSUMS_AGC_max_gain**, gives the AGC high threshold value. This is the threshold for which inputs above it produce output level at

the **CSUMS_AGC_k_Def** level and inputs with a level below it produce outputs which linearly decrease with the input level.

CSUMS_AGC_max_gain is defined as: $((10^{((\text{maximum gain in dB})/20)})/32) * 2^{23}$. Multiplying by 2^{23} converts the value into a linear 24-bit value that accommodates the 24-bit DSPs used on the DM3 boards.

Values: 0x040000 to 0x7E7DB9 (0 dB to 30 dB)

Default Value: 0x0C17E6 (9.6 dB)

Guidelines: It is recommended that the value be set in the range of 0 dB to 30 dB.

Here is a sample calculation to obtain a hexadecimal value of **CSUMS_AGC_max_gain** for a maximum gain of 21 dB:

$((10^{(21/20)})/32) * 2^{23} = 0x2CE178$

1.25.2 Documentation

The online bookshelf provided with System Release 6.0 CompactPCI Feature Pack 1 for Windows contains information about all system release features including features for application development, configuration, administration, and diagnostics.

For more information about the DCB API, see the following documents:

- *Audio Conferencing API Programming Guide*
- *Audio Conferencing API Library Reference*

For more information about modifying CONFIG files and for background information on ACG, see:

- *Intel NetStructure for DM3 Architecture for CompactPCI on Windows Configuration Guide*

Release Issues

The table below lists issues that can affect the hardware and software supported in Intel® Dialogic® System Release 6.0 CompactPCI* Feature Pack 1 for Windows*. The following information is provided for each issue:

Issue Type

This classifies the type of release issue based on its effect on users and its disposition:

- Known – A minor hardware or software issue. This category includes interoperability issues (i.e., issues relating to combining different Intel telecom products in the same system) and compatibility issues (i.e., issues that affect the use of Intel telecom products in with third-party software or hardware). Known issues are still open but may or may not be fixed in the future.
- Known (permanent) – A known hardware or software issue or limitation that will not be fixed in the future.
- Resolved – A hardware or software issue that was resolved (usually either fixed or documented) in this release.

Defect No.

A unique identification number that is used to track each issue reported via a formal Change Control System. Additional information on defects may be available via the Defect Tracking tool at http://membersresource.dialogic.com/defects/.
Note that when you select this link, you will be asked to either LOGIN or JOIN.

PTR No.

Number from problem tracking system used prior to March 27, 2006. For customer convenience, both the PTR number and the corresponding defect number are shown. For issues reported after March 27, 2006, this column contains “--” and only the defect number is used to track the issue.

SU No.

For defects that were resolved in a Service Update, indicates the Service Update number. For defects that were resolved when the base release was generally available (before any Service Updates), a “--” is shown. For non-resolved issues, this information is left blank.

Product or Component

The product or component to which the issue relates, typically one of the following:

- A system-level component; for example, Host Admin
- A hardware product; for example, Intel NetStructure DM/V Boards
- A software product; for example, the Global Call library

Description

A summary description of the issue. For non-resolved issues, a workaround is included when available.

The following table lists all issues that relate to this release, sorted by Issue Type. For other sort orders, use the following links:

- View issues sorted by Service Update Number
- View issues sorted by Product or Component
- View issues sorted by Defect Number

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Known	IPY00009092	33439		DM/IP Boards	When running on DM/IP board, system is seeing 6-8% CallP Fax test failures. All the failures are with G.711 30 ms coder. Other coders and frame sizes work fine.
Known	IPY00008110	33154		DMT160TEC Boards	When running at high density with 3 or more DMT160TEC boards and with IDD of less than 100 ms, sometimes one of the digits is missed while running digit detection with a string of 31 repeating digits.
Known	IPY00007792	32116		DMT160TEC Boards	For the DMT160TEC board, when there are 8 or more trunks in one NFAS group and calls are made on every channel at a very fast rate, the firmware may start to miss events. ***Workaround:*** Increase the value of the ISDN protocol variant parameter DisconnectTimeout in the [CHP] section of the CONFIG file to at least 3000 ms.
Known				High Availability Demo for ZT5084/ZT5550C	The RSSManager5084 demo does not get the host status; it shows the status of each SBC as the ACTIVE host. ***Workaround:*** Use the RHManager5085 demo, which also supports the ZT5084/ZT5550C.
Known	IPY00010092	33119		Host Admin	When uninstalling the System Release software, ensure that no other machines are connected through DCM remotely. Blue screen occurs during uninstall when DCM is still open on a remote machine. ***Workaround:*** Make sure that remote DCM is closed prior to uninstalling the System Release software.
Known				Host Install	The PDKManager tool, which downloads Global Call protocol modules and country dependent parameters to DM3 boards, can be set up to run automatically when DCM is started. However, after performing an update install, PDKManager no longer runs automatically. ***Workaround:*** PDKManager must be rerun manually after an update install. For further information about PDKManager, see the *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide*.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Known				Host Install	Board auto detect does not work correctly during install. ***Workaround:*** After you install the system release software and reboot the chassis or SBC, you will see “New Hardware Found” followed by identifying appropriate “Intel NetStructure ...” boards. Launch the configuration manager (DCM). If you don't see all the boards detected in DCM do the following: 1. Close DCM. 2. Go the Device Manager right click on My Computer, select Manage, go to Device Manager section). You might see one of the following: • Under Intel NetStructure DM/HDSI, some boards might show as PCI Device with a yellow ! Double-click on this device and go to re-install drivers, provide the path to \Program Files\Dialogic\Drvr, and pick dm3_wdm.inf file. Click OK. • Under Intel NetStructure IPT Board Series, some boards might show as PCI Device with a yellow ! Double-click on this device and go to re-install drivers, provide the path to \Program Files\Dialogic\Drvr, and pick pmac_wdm.inf file. Click OK. • Sometimes the boards might not be listed anywhere under Intel NetStructure IPT Board Series or Intel NetStructure DM/HDSI); they will be displayed under Other PCI Devices. Check the slot number and see if it is an Intel Dialogic board. If yes, double-click on this device and go to re-install drivers, provide the path to \Program Files\Dialogic\Drvr, and click OK. 3. Once you do this for all the devices, reboot the chassis.
Known	IPY00032702	--		Infrastructure Library	The RTF logs have about 300 prints/second. The error message reads ‘**sr_waitsync( )** timed out for thread xxxx’. You can ignore this message.
Known	IPY00024125	32289		IPT Boards	When using G.723 coders and SIP/RFC2833 with 30 ms frame size, tests that are run over high density IPT board report high number of missing digit failures.
Known	IPY00024124	32288		IPT Boards	When using IPT Series board, VAD is not supported for G.729 coder in systems greater than 1500 channels.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Known	IPY00006004	33488		IPT Boards	IPMEV_ERROR occurs during high density runs on single IPT Series board. ***Workaround:*** The channel should be dropped and reset linedev called when an IPMEV_ERROR is received on a specific channel. This will allow the channel to be re-opened.
Known	IPY00005986	32154		IPT Boards	When attempting to set PARMCH_RFC2833REDLEVEL on IPT Series products, **ipm_setparm( )** returns an EIPM_INTERNAL_INIT error code. It should return an EIPM_UNSUPPORTED error code. Changing this parameter is not supported on IPT Series products.
Known (permanent)	IPY00009256	32087		DM/IP Boards	Running Global Call over IP, applications should not open and close the channels multiple times. The channels should be opened during initialization and should remain open throughout the application.
Known (permanent)				DM/IP Boards	When an application on H.323 is using **gc_Extension( )** to extract information from a GCEV_OFFERED event, the application must ensure that it acknowledges the call within 8 seconds to prevent the offering side from timing out. The timer can be extended by sending PROCEEDING (by calling **gc_CallAck( )**) or ALERTING (by calling **gc_AcceptCall( )**) before extracting the information.
Known (permanent)				DM/IP Boards	In some cases, an E1 DualSpan DM/IP board reports 1% to 2% CSP related failures when G.729ab coders are used to establish IP calls. Most of the failures are caused due to missing TEC_STREAM event.
Known (permanent)				DM/IP Boards	While dialing/receiving MF digits over DM/IP board with G.711 coders, around 10% missing/repeat digit failures is reported. 30-40% failure rate is seen with LBR coders. MF is not supported on DM/IP board wherever MF support is discussed.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Known (permanent)	IPY00029958	36722		DM3 Drivers	DCM may not detect all boards in systems with D865GBF series motherboard. This problem occurs only with some revisions of the BIOS. ***Workaround:*** This problem can be corrected by updating the BIOS to version **P25**. First, determine if your system has the D865GBF series motherboard and check the BIOS version as follows: • From the Windows Start menu, select Run, type MSinfo32, and click OK. • In the System Information window that is displayed, check the System Model and BIOS Version/Date values. For systems with a D865GBF series motherboard, the System Model is D865GBF. The BIOS Version/Date will be something similar to this: BF86510A.86A.0075.P24. (This shows BIOS version **P24**.) If your system has the D865GBF series motherboard and an earlier BIOS than version P25, update the BIOS to version P25 as follows: • Go to the following website for the Intel Desktop Board D865GBF: *http://downloadfinder.intel.com/scripts-df-external/Product_Filter.aspx?ProductID=948&lang=eng* Follow the instructions provided at that website. Be sure to read the Release Notes and special instructions to be followed prior to installation.
Known (permanent)				DMT160TEC Boards	The Global Call Protocols documentation says that up to eight different protocols can be downloaded to the 16 spans of each DMT160TEC board. However, when using the System Release software, only five different protocols can be downloaded to the 16 spans of each DMT160TEC board.
Known (permanent)				DMT160TEC Boards	The Finland protocol is unable to run back to back. Finland protocol gets stuck when ANI is not transmitted when the default value is used for CDP_ANI_WITHAC_FACILITY_ENABLED.
Known (permanent)	IPY00009389	32849		DMV, DMV/A, DM/IP, and VFN Fax Boards with Network Interfaces	DMV, DMV/A, DM/IP and VFN Fax boards with network interfaces can route up to 256 (receive) timeslots externally to the CT Bus which can be a limitation in certain applications. For details on application development rules and guidelines regarding sharing of timeslots (SOT), see the technical note posted on the Telecom Support Resources website at: *http://resource.intel.com/telecom/support/tnotes/tnbyos/2000/tn043.htm* **Note:** DM/V-B boards, which are currently available, do not have this limitation.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Known (permanent)	IPY00006578	34616		Host Admin	Due to security enhancements implemented in Windows Server 2003 SP1, Remote DCM will no longer work with this operating system. ***Workaround:*** In order to allow Remote DCM to work again, you have to revert the security settings to the pre-service pack states for the machine being accessed remotely by modifying the following two Windows settings: First Setting: • Go to Control Panel -> Administrative Tools -> Component Services. • Go into the Properties page of Console Root -> Computers -> My Computer. • Under the COM Security tab, click on Edit Limits… button for both Access Permissions as well as Launch and Activation Permissions. • For Access Permissions, make sure "ANONYMOUS LOGON" has local as well as remote access. • For Launch and Activation Permissions, make sure "Everyone" has all local as well as remote permissions. Second Setting: • Create/modify the registry value "HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows NT\RPC\RestrictRemoteClients". It is a DWORD value that has to be set to 0 in order for Remote DCM to work. After these settings are applied, reboot the machine and the machine should be ready to be remotely managed through DCM again. Please refer to the Microsoft support website for additional information on the security enhancements in new service packs.
Known (permanent)				Host Admin	The Group One Clock Rate parameter in DCM only supports 8 MHz. If a selection of either 2 MHz or 4 MHz is made, an exception will be thrown. If this occurs, change the setting back to 8 MHz.
Known (permanent)				Host Admin	DM3 boards only allow the NETREF parameter to be set to yes. Attempting to set NETREF to a value other than yes will result in an error message when attempting to start the board with DCM. This error message will indicate that an exception has been thrown. If the value has been changed to no and the board start fails, restart DCM and change the parameter.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Known (permanent)				Host Admin	DM3 boards only support H.110 bus mode. Attempting to set the bus to a value other than H.110 will result in an error message when attempting to start the board with DCM. This error message will indicate that an exception has been thrown.
Known (permanent)	IPY00028484	36016		Host Install	When a board is first detected, its PCI bus and slot numbers are used in the unique ID for the board. If the board is moved, the unique name is kept and no longer reflects the current bus and slot numbers. Moreover, if a new board is now placed in a slot first used by another board, the new board can have the same bus and slot number in its unique name. ***Workaround:*** Do not rely on the device name displayed on the DCM GUI for getting PCI slot and bus number information. Rather, look at the "Physical" page for that board to get the current information. Also, to get this information programmatically, **NCM_GetValueEx( )** can be used to query these parameters to get the current value.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Known (permanent)	IPY00028358	33991		Host Install	After installation, the operating system does not have all the drivers loaded for all the boards. ***Workaround:*** Take the following actions. **Use Case 1:** When users see the New Hardware Wizard pop-up: 1. Click “Next”. 2. Go to the “Advanced option”. 3. Point to the location where “Intel-Dialogic” software is installed, “…\dialogic\driver\” directory. 4. Click “Next”. 5. Click “Finish”. 6. Repeat this process for all the boards for which you see the pop-up. **Use Case 2:** Installation is complete and system rebooted; not all boards are detected by DCM: 1. Go to the “Device Manager”. 2. Expand the “Intel NetStructure DM/HDSI” and see if any board has a Yellow “!”. If yes, continue with the following steps. 3. Double click on this device. 4. Click update driver / reinstall driver. 5. Follow the same steps as above (Use Case 1). **Use Case 3:** Installation is complete and system rebooted; not all boards are detected by DCM: 1. Go to the “Device Manager”. 2. Expand the “PCI Devices” and see if any board has “Unknown PCI Device”. If yes, continue with the following steps: 3. Double click on this device. 4. Click update driver / reinstall driver. 5. Follow the same steps as above (Use Case 1).
Known (permanent)				IPT Boards	When using the IPT Series board, there is a limit on the number of DTMF digits that the board can detect, process and send up to the host in one second. Attempts to exceed this limit result in failure to detect the DTMF digits using the **ipm_ReceiveDigits( )** function. Tests have indicated that the limit is approximately 350 digits per second. In other words, in one second an IPT Series board can handle 22 channels, each passing up 16 digits to the host. To handle 480 channels, each passing up 16 digits, requires a total of 20 seconds.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description

Known (permanent)				IPT Boards	When using the IPT Series board in RFC2833 DTMF transfer mode, the minimum digit-off time (time between digits) is 100 ms. (Note: A minimum of 100 ms on and off times is recommended by the RFC2833 specification). In tests with other products that support RFC2833, digit distortion was observed when using digit-off times lower than 100 ms. Since the DTMF tones are handled by another source (such as a board with tone resources), the DTMF tone definitions provided by that source must be redefined to incorporate the 100 ms inter-digit time.
					For example, if a DM/V board is providing the tone resources, parameters in the PCD and FCD files can be changed to redefine the tones as described in the following procedure:
					1. Open the PCD file corresponding to the board providing the tone resources.
					2. Search for the [COMP tonegen] section and set the InitOption parameter to No (if not already set to No).
					3. Open the CONFIG file corresponding to the board providing the tone resources.
					4. Copy the [tonegen] section into the CONFIG file.
					5. Note the values for the digit on time and digit off time. The units for these values are samples, where 8 samples = 1 ms. If the desired ms value is known, simply multiply that value by 8 to get the value in samples.
					`ToneDesc SegOnDur 1 800 !! Currently set to 100 ms...(100x8)=800 samples`
					`ToneDesc SegOffDur 1 800 !! Currently set to 100 ms...(100x8)=800 samples`
					6. Run the fcdgen utility on the CONFIG file to generate the corresponding FCD file; for example:
					`fcdgen qs_t1.config`
					The FCD file generated in this case is *qs_t1.fcd*.
					7. Use the new PCD file and FCD files when downloading the firmware to the board containing the tone resources.
					The [tonegen] section for the CONFIG file is available separately below the Release Issues table.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Known (permanent)				IPT Boards	When using the Global Call IP call control library with SIP and running a large number of channels (e.g. 1560), the application may not always receive GCEV_PROCEEDING before GCEV_CONNECTED. Network conditions may cause UDP packets to be lost, some of which may contain the SIP “100 Trying” response message that corresponds to the GCEV_PROCEEDING event. And because SIP does not provide for retransmission of 1xx response messages, no GCEV_PROCEEDING event will occur when a UDP packet containing a “100 Trying” is lost. ***Workaround:*** Applications should be written to continue processing the call when GCEV_CONNECTED is received regardless of whether a GCEV_PROCEEDING was received.
Resolved	IPY00037356	--	75	Board Detection	DCM assigns the same physical slot ID to two boards (in different physical slots).
Resolved	IPY00033640	--	73	Board Detection	When installing a board in an active system (hot insertion), an error appears in DCM and in DebugAngel for a dummy board, and the insertion fails.
Resolved	IPY00031561	36755	68	Board Detection	Intermittent blue screens appear when shutting down or rebooting OS with boards downloaded.
Resolved	IPY00038074	--	77	Board Download	The OAMSYSLOG component reports multiple “DM3FDSP - GetOverlappedResult()[2] timeout for board 5, Error= 121” entries in RTF logs during load test.
Resolved	IPY00036025	--	73	Board Download	Dr. Watson crash occurs during board detection/download on a Motorola cPCI chassis.
Resolved	IPY00037989	--	76	Call Control	The CLI cannot be obtained using either **gc_GetCallInfo( )** or **gc_GetANI( )** when the screening indicator bits are 00 (user provided) or 01 (verified & passed).
Resolved	IPY00037864	--	76	Call Control	When using DMN160TEC board, outgoing ISDN calls are failing with a disconnect cause of 0xE1.
Resolved	IPY00028547	35670	54	Call Control	PDK protocol delivers DETECTED/OFFERED event to the channel even if **gc_ReleaseCall( )** was never called to clean up the previous call on this channel. The protocol will transition to detected/offered state for a new call before the release-call of a previous call is called on the same channel. Once a new call attempts to be transmitted/received on this channel, the state machine becomes confused and returns an error. It especially affects multithreaded applications when call setup and call teardown are handled from within different contexts.
Resolved	IPY00006846	36711	54	Call Control	Crash due to corruption in PDKRT library’s internal database caused by application.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Resolved	IPY00037619	--	76	Conferencing	When an application thread calls the **dcb_estconf( )** function, lower level library threads crash (enter and exit immediately) and the application receives an access violation error.
Resolved	IPY00009145	33778	8	DCM	During a failover (going from an active domain to another), DCM fails to save all the board information in a timely manner so that it could be used after the failover. As a result, when the failover occurs, the boards show up in DCM as disabled. **Note:** For information about functions that have been added to the NCM API to handle this issue, see Section 3.3.9, "Native Configuration Management API Library Reference", on page 106 in the Documentation Updates chapter.
Resolved	IPY00033472	--	64	DM/IP Boards	Specifying multiple DTMF detection methods prevents fax CED detection.
Resolved	IPY00028236	33067	47	DM/IP Boards	When T.38 is run on all of the DM/IP channels, T.38 fax tests fail at a high rate with E1 DM/IP board running H.323/Out-of-band mode.
Resolved	IPY00009720	33823	8	DM/IP Boards	When attempts to register with a gatekeeper fail, the system handles increase in every attempt along with the virtual memory. The application tries to register and then it receives a TASKFAIL. Then it tries to register again in a loop. In every attempt, the handles and the virtual memory increase.
Resolved	IPY00008792	31055	--	DM/IP Boards	CPLD is not being correctly initialized on the DM/IP resource board. It causes CT Bus timeslots to be incorrectly assigned to CT Bus.
Resolved	IPY00007259	31994	--	DM/IP Boards	The ml1b_ds2_r2mf.fcd, ml10_ds2_r2mf.fcd, and ipvs_evr_2r2mf_ml11_311c.fcd files download with SRAM corruption error.
Resolved	IPY00006816	36737	64	DM/IP Boards	Player component incorrectly terminating on DTMF when DTMFs are not present.
Resolved	IPY00010200	32920	--	DM/V1200A Boards	When inserting a new voice card in the chassis, the driver does not install automatically and shows "!" signal in Device Manager.
Resolved	IPY00007822	31842	--	DM/V1200A Boards	The DM/V1200A-4E1 board fails to start when using ml1b_qs2_ts16 load.
Resolved	IPY00009058	32889	--	DM/V2400A Boards	When running DM/V2400A on high load for one day, DLGC_EVT_SP_FAILURE and DLGC_EVT_CP_FAILURE occurs.
Resolved	IPY00007825	32559	--	DM/V2400A Boards	After a fax Tx/Rx is attempted when using DM/V2400A with ML5, the board stops responding.
Resolved	IPY00028649	36416	50	DM3 Drivers	Blue screen crashes occur in dlgcmpd when using DM/V480A-2T1-cPCI boards on Windows 2003.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Resolved	IPY00030893	31998	--	DM3 Fax	The DM/V2400A board crashes after send/receive fax calls are made.
Resolved	IPY00028375	35507	36	DM3 Fax	When you implement a send fax and receive fax in one call by using turnaround polling, the polling bit is not updated when the receiving fax contains more than one page. This causes the **fx_rcvfax( )** function to complete with TM_POLLED instead of TM_FXTERM.
Resolved	IPY00010751	34310	28	DM3 Fax	When performing a loopback test for sending and receiving fax on a T1 trunk, data was lost.
Resolved	IPY00007791	32114	--	DM3 Fax	Fax send/receive tests fail on the mixed conference/fax loads.
Resolved	IPY00007389	33492	28	DM3 Fax	After test is running for some time, faxes stop being sent/received.
Resolved	IPY00028641	36296	51	DM3 Firmware	Firmware does not send GCEV_PROGRESSING event to application upon reception of a PROGRESS message with unknown but syntactically correct event. Unless the message is incorrectly formatted, the event should always be generated.
Resolved	IPY00028633	35748	51	DM3 Firmware	Sometimes noise is generated when one party leaves a conference. The noise disappears when a party is added to a conference.
Resolved	IPY00028355	32918	8	DM3 Firmware	When using a DMV960-cPCI board in with ml1_qs_ni2 firmware, the board name appears in DCM as DMT960.
Resolved	IPY00028315	35197	36	DM3 Firmware	A board assert occurs after a period of time when the application employs a quick media start and stop. Messages are printed in the DebugAngel log indicating that a media stop command was sent to the firmware before the media start command was complete. After successive events like this, the board asserts.
Resolved	IPY00028295	34503	28	DM3 Firmware	Application gets TDX_ERROR events after **dx_playiottdata( )**.
Resolved	IPY00010151	34848	18	DM3 Firmware	The D-channel fails to recover when the firmware persistent layer 2 activation feature is enabled.
Resolved	IPY00009803	34881	18	DM3 Firmware	The host application intermittently fails to receive notification of the D-channel coming up when the firmware persistent layer 2 activation feature is enabled.
Resolved	IPY00038533	--	82	DM3 Host Runtime Library	An internal parameter is not decremented correctly when a process exits, causing failures in opening devices.
Resolved	IPY00028306	35954	39	DM3 Host Runtime Library	When using a DM/V1200A board with ML9b_qs2_e1, after doing a PHS, the **dcb_estconf( )** function failed.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Resolved	IPY00016077	31146	18	DM3 Host Runtime Library	Running NI2 protocol, after several hours all channels lock up simultaneously.
Resolved	IPY00010764	35101	37	DM3 Host Runtime Library	Channels get blocked under heavy load with call collisions. This occurs when running the pdk_us_mf_io protocol, with two trunks connected back-to-back, with an application instance per trunk.
Resolved	IPY00009546	35776	39	DM3 Host Runtime Library	When running the gc_basic_call_model demo application using DMT160TEC board and PDK R2 protocol for load test, after 3-4 hours, calls cannot be connected and errors are shown in DebugAngel log,
Resolved	IPY00008007	31993	--	DM3 Host Runtime Library	The dm3cc_parm.h file is missing after installing the release with Global Call Protocols.
Resolved	IPY00007969	30577	--	DM3 Host Runtime Library	When the application issues **dt_unlisten( )** while there still is a voice device listening to it, any subsequent **dt_unlisten( )** or **dx_listen( )** fails on that device.
Resolved	IPY00034559	--	71	DM3 IP	Assert in the Radvision stack.
Resolved	IPY00028598	35797	37	DM3 IP	Any exotic fields that a SIP phone would send are copied into the response from the INVITE. For example, some equipment sends 'a=silenceSupp:off - - - -' and that gets copied.
Resolved	IPY00028439	36196	44	DM3 IP	SIP INVITE messages that contain more than 255 bytes are rejected and do not result in an OFFERED event being sent to the application.
Resolved	IPY00028382	35440	39	DM3 IP	The type field in the IP_CAPABILITY structure is set to zero (unknown) when it should be set to two (GCCAPTYPE_AUDIO).
Resolved	IPY00028223	36307	44	DM3 IP	GCEV_TASKFAILs occur if a digit is received after ReleaseCall, but before GCEV_RELEASECALL is posted.
Resolved	IPY00028221	36083	39	DM3 IP	**gc_OpenEx( )** fails on IPT1200C board after upgrading to Service Update 18.
Resolved	IPY00011022	36413	50	DM3 IP	When using Global Call to develop a VoIP/SIP based application, you can call **gc_OpenEx( )** in ASYNC mode with a GC IP based line device such as "::N_iptB1T1:P_SIP:M_ipmB1C1:V_dxxxB1C1". This will sometimes result in an internal lib error "RESULT_COMPONENT_ERROR, error code: 0x2800c" which can be found in RTF logging, even though the actual **gc_OpenEx( )** call returns a success.
Resolved	IPY00010760	36647	50	DM3 IP	When a call is placed to an IP address that does not exist or to a valid IP address that does not have a SIP phone active, you cannot call **gc_DropCall( )** to disconnect the call; you have to wait for the 64-second INVITE timer to expire before you receive a GCEV_DROPCALL.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Resolved	IPY00009693	34505	18	DM3 IP	**gc_ReqService( )** doesn't get a completion event or return a failure while performing gatekeeper registration.
Resolved	IPY00009617	35140	28	DM3 IP	TSM content field in the Annex M Message is not initialized properly before a call to the Radvision H323 stack API. This causes the Radvision stack to crash when it can not resolve this field.
Resolved	IPY00030904	36245	47	DM3 Network	**sr_getboardcnt( )** does not count ipt devices on IPT boards (such as IPT1200C, IPT10000C, etc.). “0” ipt devices is returned.
Resolved	IPY00028663	36667	51	DM3 Network	Race condition on network side of ISDN protocol causes time slots to remain out of service.
Resolved	IPY00028624	36186	51	DM3 Network	Some ISDN channels stop responding after **gc_MakeCall( )** is called for first time. This results in lost channels (usually 2-4 per span) until the board is stopped and restarted.
Resolved	IPY00016070	29869	--	DM3 Network	After re-download on the remote side, the local Layer2 still thinks that Layer1 is down and does not issue SABME request.
Resolved	IPY00010955	35106	28	DM3 Network	Application sometimes receives a GCEV_DISCONNECT message for a SETUP glare condition that has a invalid cause value.
Resolved	IPY00010232	33667	33	DM3 Network	Missing GCEV_UNBLOCKED events for channels configured for DPNSS or DASS2 in a mixed system running other ISDN protocols with NetCRV support enabled.
Resolved	IPY00010117	35138	28	DM3 Network	On some occasions, a GCEV_UNBLOCKED event is generated while or after a channel has been torn down remotely. (It is expected that a GCEV_BLOCKED event should be generated in this case.)
Resolved	IPY00009838	35051	28	DM3 Network	The firmware is not sending the OFFERED event for new incoming calls instances. The calls are not rejected, and the caller is left to time out.
Resolved	IPY00009101	32894	--	DM3 Network	NFAS trunks could receive GCEV_OFFERED but could not accept or answer the call on DM/V960A-4T1 and DM/V960-4T1.
Resolved	IPY00008295	29037	--	DM3 Network	When using ml2_qs2_net5 firmware with DM/V1200A, if CRC is disabled in the fcd file, when the board is restarted, the D-channel is not active and no call can be made.
Resolved	IPY00007988	31199	--	DM3 Network	D-channel will not recover if the line is taken out-of-service and put back in-service or if the physical line is unplugged and plugged back in several times.
Resolved	IPY00007866	31673	--	DM3 Network	The DM/V1200-4E1 board fails to start when using ml2_qs2_ts16 load.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Resolved	IPY00010784	34891	18	DM3 Tools	PDKManager fails to download CAS protocol.
Resolved	IPY00009266	34050	18	DM3 Tools	The dlgagent.log file under “c:” is created by SNMP automatically when DCM starts. When SNMP cannot get the status of boards, the log records the events. After running an application for several days, the size of the log increased to 52M.
Resolved	IPY00028601	35250	36	DM3 Voice	E1 cPCI boards can be started with mu-law, but DTMF could not be detected.
Resolved	IPY00009683	33685	28	DM3 Voice	**dx_stopch( )** cannot stop voice channel if run in thread.
Resolved	IPY00009433	34878	33	DM3 Voice	**dx_playiottdata( )** ignores the data length and plays until EoF, which sometimes causes noise if there is additional data after “data chunk.”
Resolved	IPY00008760	32598	--	DM3 Voice	The play/record ADPCM tests causes firmware crash on ml1b_qsa load when running max channels.
Resolved	IPY00007872	33351	28	DM3 Voice	When calling **dx_playtoneEx( )** with DX_MAXTIME longer than time to play 40 repetitions of the tone defined, the function terminates. **dx_playtoneEx( )** should play as long as defined, not only 40 repetitions.
Resolved	IPY00007687	31999	--	DM3 Voice	The DM/V2400A board crashes after 4 channel CSP starts.
Resolved	IPY00009533	33419	--	DMN160TEC Boards	When calling **gc_MakeCall( )** causes a SETUP message to be sent out, if the first response form the other side is CONNECTED, the board responds with CONNECT_ACK but GCEV_CONNECTED is not sent up to the application.
Resolved	IPY00009045	32758	--	DMN160TEC Boards	Call cannot be made on DMN160TEC board since the data link is not correctly established.
Resolved	IPY00009034	31705	--	DMN160TEC Boards	When two DMN160TEC boards are installed and E1 connected, DCM could not be downloaded,
Resolved	IPY00008938	32384	33	DMN160TEC Boards	A single way voice issue occurs on DMN160TEC (dti16_e1cc.fcd). The interface (such as dtiB1T2) on the DMN160TEC could not hear any other timeslot (including the network interface or resource on other board, or any interface on the same board), but the network interface or resource on other board could hear the interface on DMN160TEC.
Resolved	IPY00007916	32554	--	DMN160TEC Boards	When the span is set as NET5 Network End and an incoming SETUP comes in without a Channel ID IE, the SETUP is ignored.
Resolved	IPY00007245	30686	--	DMN160TEC Boards	If Layer 3 fails on a single span of a DMN160TEC board, all other spans are disabled and no calls can be handled.
Resolved	IPY00034841	--	71	Global Call	H.323 channels stuck in Indeterminate state.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Resolved	IPY00028530	36371	47	Global Call	When **gc_Start( )** fails, **gc_ErrorInfo( )** cannot be used to retrieve the error code. The **gc_ErrorInfo( )** function fails with an error indicating that **gc_Start( )** has not been issued. The correct behavior is for **gc_ErrorInfo( )** to execute successfully and return the error code and description.
Resolved	IPY00028207	36310	44	Global Call	**gc_CompleteTransfer( )** does not complete successfully. Error returns: “Function not supported in current state.”
Resolved	IPY00006790	35137	68	Global Call	For outbound GC SS7 calls with dialstring *1234, the leading * is stripped and replaced with a trailing 0 (i.e., 12340) causing call to fail.
Resolved	IPY00006782	34038	8	Global Call	gc_lib is not registering board devices fast enough; the impact of this causes **gc_GetMetaEvent( )** to return NULL for GCEV_D_CHAN_STATUS event(ldev:0, crn:0x0h).
Resolved	IPY00006654	36085	41	Global Call	Using Global Call SS7 protocol, when ISUP sent SUSPEND and RESUME message, the Global Call library did not generate a GCEV_RETRIEVECALL event.
Resolved	IPY00010543	35520	37	Global Call IP	When using H.323, calling **gc_Extension( )** with IPPARM_PHONELIST may cause an application failure.
Resolved	IPY00010486	35596	37	Global Call IP	Application cannot access the full content of the Bearer Capability IE of an inbound SETUP message.
Resolved	IPY00009870	35585	37	Global Call IP	Secondary Call Reference Number (CRN) is not provided on party C in a case of attended SIP transfer.
Resolved	IPY00007960	35693	37	Global Call IP	Gatekeeper registration after un-registration fails.
Resolved	IPY00007220	35696	37	Global Call IP	Application fails to start when executed as a network service. When executed from command line, it starts fine.
Resolved	IPY00039163	--	82	H.323 Call Control	**gc_UnListen( )** fails in unanswered H.323 call.
Resolved	IPY00037372	--	73	H.323 Call Control	An access violation/assert is seen in the Global Call IP Call Control library if a RequestMode message for changing audio codecs is received.
Resolved	IPY00037351	--	73	H.323 Call Control	When the remote capabilities contain one audio codec and T.38 fax codec, the Global Call IP Call Control Library will incorrectly attempt to switch to fax.
Resolved	IPY00008337	33011	18	HDSI/1200	Changes made to the ring frequency parameter via the FCD file do not take effect.
Resolved	IPY00033164	--	71	Host Admin	Lost Packet Alarm not working on IPT Series boards.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Resolved	IPY00032041	28707	--	Host Admin	DCM keeps running and fails to download if there is network traffic on the ISDN spans during download.
Resolved	IPY00031597	36527	59	Host Admin	Autodump leaves the board in a limbo state when it fails to download diagnostic firmware.
Resolved	IPY00030903	32316	50	Host Admin	**NCM_GetVersionInfo( )** reports incorrect values for the DSS version information.
Resolved	IPY00028447	36318	45	Host Admin	There is a problem with RSS operation on the ZT5084/ZT5550 chassis and SBC. After a forced switchover, DCM does not display the voice board (DM/V1200A) in the active SBC.
Resolved	IPY00023859	32197	--	Host Admin	The DM/F300 board is downloaded with SRAM corruption error.
Resolved	IPY00023758	32196	--	Host Admin	The HDSI/1200 board is downloaded with SRAM corruption error.
Resolved	IPY00010697	34874	28	Host Admin	There is a memory leak when using SNMP service with DM/IP601-cPCI board.
Resolved	IPY00010536	34988	28	Host Admin	**NCM_SaveConfiguration( )** and **NCM_RestoreConfiguration( )** return success even if a failure has occurred. An error message is generated in the log file, but the functions still return success.
Resolved	IPY00010485	35110	28	Host Admin	When running DCM on Windows 2003 Server, the detection process ends abnormally with errors. This is due to the dcmobj service terminating unexpectedly.
Resolved	IPY00009438	33687	18	Host Admin	When using snmptrapd -P on DM/V960-4T1-cPCI to catch hardware alarms, the “detectedNoAlarm (100)” cannot be received at times when the cable is unplugged and plugged back in.
Resolved	IPY00009321	33102	--	Host Admin	If CDTNetworkEventHandler listening is started on its own, it will stop listening when asked. If CDTNetworkEventHandler listening is started with all the other channels, stopping listening on the network channel hangs.
Resolved	IPY00009263	33385	28	Host Admin	When a DM/V-A or DM/V-B board is shut down in DCM, an error event is generated in the Windows event viewer. The error message is “dwCheckPoint=6”.
Resolved	IPY00009168	29079	--	Host Admin	When trying to modify the DefaultJitterBufferLength value for IPT Series board in DCM, if the value is greater then 15, the value will return to the default value of 3. This is true even if the value has been modified when the board was started.
Resolved	IPY00008919	32461	--	Host Admin	When setting up with Windows 2003 Chinese Simplified version, DCM connect shows genload error.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Resolved	IPY00008881	33156	8	Host Admin	Problems with time slot assignments for devices dtiB1T1 to dtiB1T10 occurred on the following boards: DMN1200-4E1-cPCI (4X0_ts16.fcd) and DMN160TEC (dti16_e1cc.fcd).
Resolved	IPY00008697	32202	--	Host Admin	Event log reports that the timeslot could not be releases with DM/V2400A when stopping DCM.
Resolved	IPY00008689	35188	33	Host Admin	The fault detector, which checks whether a board is running as expected, was sending too many messages, causing increased CPU utilization at the host and kernel level.
Resolved	IPY00008673	32460	--	Host Admin	When setting up with Windows 2003 Chinese Simplified version, the win device manager does not set up the voice card driver correctly.
Resolved	IPY00008503	33686	8	Host Admin	When using snmptrapd -P on DMV960-4T1-cPCI and DMN160TEC boards to catch hardware alarms, the lineindex is not stable when the cable is unplugged and plugged back in.
Resolved	IPY00008483	28628	28	Host Admin	During uninstall, some files related to IPT672 may remain in the ..\Dialogic\CFG directory upon completion.
Resolved	IPY00007845	29538	--	Host Admin	Third party client support is mentioned in the documentation, but the tools are not present in the release.
Resolved	IPY00007441	28892	33	Host Admin	The **sr_getboardcnt( )** function does not get the correct board count when using ml9b firmware for DM/V1200A boards, or ml9b / ml9c firmware for DM/V2400A boards. The board count shows as 0.
Resolved	IPY00032043	28539	--	Host Drivers	The DM3E standard error tool will cause a blue screen when you exit the program while a DMT160TEC board is downloaded.
Resolved	IPY00007412	32005	18	Host Drivers	During CPU failover, the primary SBC hangs while the secondary SBC requests for the domain switchover and takes over, due to the application having open handles to the devices (Intel boards) and not getting query_remove or remove events.
Resolved	IPY00030892	35704	36	Host Install	The cleanup utility does not remove the IPMedia service.
Resolved	IPY00030888	35705	36	Host Install	There are some hard-coded path entries in dlgc_rel_clean.bat (the cleanup utility), and this causes installation problems.
Resolved	IPY00028464	34945	33	Host Install	Attempting to install Service Update 8 on a system that already has Feature Pack 1 installed (but not the SDK), cannot be completed.
Resolved	IPY00009988	32756	--	Host Install	When DCM starts, an error message box appears regarding Springware boards.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Resolved	IPY00007401	30072	--	Host Install	If a system has 2 DMN160TEC boards, the board fails to restart in DCM (board detected by DCM) after hot removing and hot re-inserting 1 DMN160TEC board.
Resolved	IPY00038849	--	82	Host Library	When opening channels asynchronously with **gc_open( )**, sequentially one after another channels fail to open.
Resolved	IPY00035831	--	71	Host Library	Segmentation fault occurs in libipm_vsc.so when calling **gc_close( )** on GlobalCall (IP based) line device.
Resolved	IPY00032039	28312	--	Host Library	**dx_get_tngencad( )** returns incorrect data for multi-segment tones.
Resolved	IPY00028542	36633	50	Host Library	Access violation occurred with **sr_putevt( )**/**gc_GetMetaEvent( )**.
Resolved	IPY00021354	28115	--	Host Library	Caller's voice cannot be heard when joining another conference without hanging up the first call.
Resolved	IPY00010556	35157	33	Host Library	Calling **ATDV_SUBDEVS( )** on DM3 MSI stations while alarms are being processed on DM3 trunks on span cards can cause a deadlock condition in the application, which can hang the system for 8 seconds.
Resolved	IPY00006712	36790	54	Host Library	No GCEV_MEDIADETECTED event is received when the first sound heard after a connect is a SIT tone (frequency 914 Hz).
Resolved	IPY00028356	32919	39	Hot Swap Kit	Invoking the "Other Chassis" menu selection during the Hot Swap Kit (HSK) installation for any chassis or SBC not in the HSK supported list is not recommended, as it can produce adverse effects to system stability.
Resolved	IPY00032262	36688	54	Infrastructure Library	When you have an application that creates two threads, each thread monitors its own events using **sr_waitevtEx( )**. The first thread makes a call, synchronously. The second thread makes a call, synchronously. Before this second call is connected the first call is disconnected. At this time you do not receive a CCEV_DISCONNECT event for the first call. The **sr_waitevtEx( )** call just hangs for the first thread.
Resolved	IPY00032265	36780	54	Infrastructure Library	Standard Runtime Library (SRL) seems to get into a Hung state causing event and IO to stop.
Resolved	IPY00038060	--	77	IP	An assert occurs when there are no media attributes containing "rtpmap" in a SIP INVITE.
Resolved	IPY00037391	--	82	IP	Access violations occur; Dr. Watson dumps indicate possible problem with **gc_GetMetaEvent( )**.
Resolved	IPY00011037	36677	53	IP Host	Inbound fax call fails. This happen when previous call on the same device is dropped and media devices are disconnected using **gc_SetUserInfo( )**.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Resolved	IPY00006077	36237	62	IP Media Session Control (RTP)	Unexpected RTCP Goodbye messages occur during call.
Resolved	IPY00041296	--	83	IPT Boards	Outgoing H.323 calls result in GCEV_TASKFAIL at the Alerting stage. After the call's release, no successful call can be made on that channel (iptB1Tx, ipmB1Cx device) until the board is reset.
Resolved	IPY00041280	--	83	IPT Boards	Dr. Watson crash occurs when attempting to make an outbound H.323 call.
Resolved	IPY00037302	--	78	IPT Boards	After 120k calls (about once every 2 days), the IPT4800 board reports GCEV_TASKFAIL, and IPMEV_ERROR appears in the RTF log.
Resolved	IPY00033005	--	59	IPT Boards	T.38 fax auto switch doesn't work with SIP.
Resolved	IPY00032954	--	59	IPT Boards	With an ISDN->H323 fax call, the call sets the codec up as G729 then resets to T.38 on fax tone detection. The fax is sent through OK, but on IPT Series board shutdown of the media session, the line device returns a GCEV_TASKFAIL and it does not respond to a reset line device. The application can no longer use that IP channel.
Resolved	IPY00028460	36298	50	IPT Boards	An extra “reserved” codec value is sent in the INVITE if coders set to “don’t care.”
Resolved	IPY00028222	36483	50	IPT Boards	IPT assert occurs under traffic.
Resolved	IPY00010208	32866	--	IPT Boards	Pressing DTMF on the PSTN side when connected via conference to IP (SIP) phone causes Radvision stack to crash.
Resolved	IPY00009510	32661	--	IPT Boards	Unknown event messages from the RTF logs.
Resolved	IPY00008911	32605	--	IPT Boards	The Line Device state machine does not handle the race condition when **gc_ResetLineDevice( )** is called while the ipm media device is in process of pending upon **ipm_Stop(STOP_MEDIA)** completion.
Resolved	IPY00008522	33604	18	IPT Boards	The IPT Series board firmware update utility invokes the pdiResetBoard function to reset the board, but the reset does not occur. The update utility tool works only on Windows 2000 and not on Windows 2003.
Resolved	IPY00008203	31987	--	IPT Boards	After running the application for several hours, a few IP channels will be blocked and cannot be recovered until re-downloading the board.
Resolved	IPY00007103	30875	--	IPT Boards	The IPT Series board with 16 DSP is detected as IPT4800 instead of IPT6720.
Resolved	IPY00009319	28981	--	Media Span Boards	During conferencing, when party A (transmit or full duplex) plays a music file and party B (full duplex) speaks at the same time, party C (full duplex) can only hear conferee B clearly. If all parties are on network devices, this problem does not appear.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Resolved	IPY00029931	36809	59	NCM API	Unable to run application from debugger.
Resolved	IPY00009690	33556	8	NCM API	**NCM_GetFamilyDeviceByAUID( )** return failed, causing a “memory can not read “error message.
Resolved	IPY00038280	--	82	OA&M	A non-OAMIPC based client was attempting connection to an internal software component, an OAMIPC-based server. This caused the internal OAMIPC-based server to crash when invoking the Dialogic® system service startup or shutdown.
Resolved	IPY00032996	--	55	OA&M	Fixed internally found bugs in OA&M clean-up utility.
Resolved	IPY00032271	36699	59	OA&M	There is a limitation to the amount of processes you can use because of a limitation of signals you can create in the operating system.
Resolved	IPY00028495	34180	47	Protocols	With a DM/V960-4T1-cPCI board looped to a DMS switch, the switch sends “Maintenance” message(Interface ID Present = 0) to the DM/V960 - DMS side, and the DM/V960 - DMS side still returns 4 octets “Maintenance Ack” (Interface ID Present = 1) to switch. The “Maintenance Ack” (Interface ID Present = 1) is denied by the switch.
Resolved	IPY00028378	34586	33	Protocols	For inbound call, channel is blocked after the remote caller hangs up before sending DNIS, when using pdk_hk_dtmf_io.cdp.
Resolved	IPY00028243	33496	42	Protocols	ISDN answer call fails on a high density system. The error is “the function is not supported in this state.”
Resolved	IPY00010746	35042	33	Protocols	When using the pdk_us_mf_io protocol, if CDP_OUT_Send_Alerting_After_Dialing = 1 and CPA is disabled, the user expects to get the GCEV_ALERTING event right after dialing. However, if the remote side answers the call too quickly, the GCEV_CONNECTED event is returned and the GCEV_ALERTING event never comes in.
Resolved	IPY00010621	34537	33	Protocols	When using the pdk_us_mf_io protocol in the Feature Group D configuration, ANI is missing the last digit when ANI is not terminated with the expected ST digit.
Resolved	IPY00010372	35035	33	Protocols	After sending CAS_HOOKFLASH, there should be some delay before sending DTMF in pdk_sw_e1_necls_io protocol, if CDP_WaitDialToneEnabled = 0 (i.e., do not wait for dialtone).
Resolved	IPY00010223	34985	33	Protocols	pdk_sw_e1_ermx_io.cdp can only accept one ringing signal (the internal ringing or the external ringing but not both). Defining CAS_RING_APPLIED (0001 -> 0xxx) solves the detection of the two ringing signals but causes problems with outgoing calls.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Resolved	IPY00010129	34274	33	Protocols	Global Call does not provide a way to disable DISCONNECT TONE SUPERVISION with pdk_na_an_io.cdp.
Resolved	IPY00010035	35159	33	Protocols	Under certain conditions when a **gc_MakeCall( )** attempt times out, it incorrectly displays the result message as NORMAL CLEARING instead of timeout.
Resolved	IPY00010004	34685	33	Protocols	When using the pdk_us_mf_io protocol in the Feature Group D configuration, the protocol does not send a Disconnect signal when it times out waiting for DNIS and ANI. This occurs when the remote side is configured as Feature Group B and makes a call.
Resolved	IPY00009837	35049	33	Protocols	There seems to be a hard-coded 30-second timeout on a Make Call when the call is made in Alerting mode, which will terminate the call if no one picks up the phone. The expected behavior is that the call will not be dropped automatically, so the phone will ring forever if no one picks up. This occurs on T1 CAS lines.
Resolved	IPY00009409	34663	33	Protocols	When using FXS protocol and calling a busy station using supervised transfer, you get a disconnect event for both the consultation CRN and transferred CRN.
Resolved	IPY00008220	34972	33	Protocols	When using the pdk_us_mf_io protocol, digits from the previous call are returned in ANI.
Resolved	IPY00008218	29626	--	Protocols	When using ml2c_ds2_r2mf firmware with Korean R2 protocol on DM/V600A-2E1, the call cannot connect.
Resolved	IPY00007838	28948	--	Protocols	gc_basic_call_model failed to run with pdk_cn_r2_io protocol.
Resolved	IPY00007685	30541	--	Protocols	**gc_GetDNIS( )** doesn't retrieve the second part of the DNIS using pdk_ccitt_r2_io protocol.
Resolved	IPY00007327	30233	33	Protocols	With the pdk_mx_r2_io protocol, if the E1 cable is disconnected and reconnected, the application does not receive all the GCEV_UNBLOCKED events.
Resolved	IPY00006811	36584	50	Protocols	The pdk_us_ls_fxo protocol fails to notify the PDK library that the disconnected call has been already released, which prevents the application from dropping the call when a new incoming call is pending.
Resolved	IPY00006809	34543	33	Protocols	When CDP_IN_DNIS_ST_Needed = 0, the pdk_e1_cas_io protocol should not issue timed-out error while waiting for DNIS.
Resolved	IPY00006804	34319	33	Protocols	If a board is configured using *pdk_us_ls_fxs_io.cdp* file and a call is abandoned after the first ring, the application is not receiving the GCEV_DISCONNECTED event that is expected.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Resolved	IPY00006771	34329	33	Protocols	Using Belgium R2 protocol, when configured in DTMF/MF mode, in the Offered state the **gc_ResetLineDev( )** function does not behave properly.
Resolved	IPY00006762	34664	33	Protocols	When using E1 line side protocol and calling a busy station using supervised transfer, you get a disconnect event for both the consultation CRN and transferred CRN.
Resolved	IPY00006755	33780	51	Protocols	With 5ESS ISDN on DM3, call setup fails when the CALLED NUMBER TYPE is set to NETWORK_SPECIFIC (0x03).
Resolved	IPY00006748	34587	33	Protocols	The PDK E1 CAS protocol cannot be downloaded on DM3 boards, and Springware board channels cannot be opened when using this protocol.
Resolved	IPY00038979	--	82	PSTN Call Control	The pdk_sw_e1_fxs_io protocol does not forward the correct reason when a call is disconnected due to detection of a SIT. The reason should indicate that SIT was detected.
Resolved	IPY00036248	--	75	PSTN Call Control	When using Global Call SS7 the 0xb and 0xc address signals, which were previously reported to the app as “b” and “c”, are now getting reported as “#” and “*”, thus breaking backward compatibility.
Resolved	IPY00035451	--	75	PSTN Call Control	**gc_OpenEx( )** fails for device “:N_dkB1T1” for SS7 board when configured for clear channel.
Resolved	IPY00035148	--	75	PSTN Call Control	The **gc_Unlisten( )** function has no effect when issued on “dk” devices using Global Call SS7.
Resolved	IPY00034816	--	75	PSTN Call Control	SIT tone not detected on Nortel Meridian protocol.
Resolved	IPY00034429	--	75	PSTN Call Control	Unable to detect RINGBACK tone on outbound calls using Venezuela DTMF signaling.
Resolved	IPY00033698	--	68	PSTN Call Control	The primary call can not be re-transferred via **gc_SetupTransfer( )** when the transferred call is disconnected after SwapHold.
Resolved	IPY00037432	--	75	Runtime Trace Facility (RTF)	The **dx_clrdigbuf( )** function overwrites area of thread’s stack space, causing the application to crash.
Resolved	IPY00036919	--	75	Runtime Trace Facility (RTF)	Unable to configure RTF trace capabilities using RTFManager because the selection is grayed out.
Resolved	IPY00038956	--	82	SIP Call Control	The SDP in a SIP “200 OK” message contains “a=” lines that don't correspond with the “m=” line.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Resolved	IPY00038572	--	82	SIP Call Control	When running a Dialogic® Global Call IP-based application that enables notification of SIP messages through GCEV_EXTENSION events, the type of SIP message received with the event cannot be identified. The message type value retrievable with that event returns more bytes than expected, making it unable to decipher which message was received.
Resolved	IPY00038365	--	78	SIP Call Control	Egress SIP calls work briefly, but then omit SDP in egress INVITE message.
Resolved	IPY00034627	--	71	SIP Call Control	Format Specific Parameters of the Media Format Attribute are not sent within SDP body of an Invite message when using non-default value for IPPARM_DTMF_RFC2833_PAYLOAD_TYPE.
Resolved	IPY00034406	--	71	SIP Call Control	The handle count of the application is very high.
Resolved	IPY00033763	--	64	SIP Call Control	IPT Series boards incorrectly timestamping packets.
Resolved	IPY00036855	--	77	SNMP	When using MIB2 from RFC1213, Dialogic SNMP agent fails to return valid information when a “get” command is issued.
Resolved	IPY00028383	35321	33	Springware Voice	Busy tones are detected as “no ringback” in call progress analysis when using the **dx_dial( )** method in Global Call application.
Resolved	IPY00037918	--	77	SS7	The RSI link goes down intermittently.
Resolved	IPY00037767	--	77	SS7	The GCSS7 library does not generate the GCEV_MOREINFO event if it receives a SAM message with only STOP digit (0xf) after the application has already issued **gc_CallAck( )**.
Resolved	IPY00037632	--	77	SS7	If there is a delay in the SS7 server picking up messages from the IPC queue, an ERROR_IO_PENDING occurs and the SS7 library terminates the IPC. This causes all the circuits to get blocked, as there is no more connection with the SS7 service. This is causing the IVRs to get a sudden circuit block from the switch in all of its SS7 circuits.
Resolved	IPY00034404	--	68	SS7	In Global Call SS7, initial alarm conditions are not progagated up to application.
Resolved	IPY00030694	31794	--	SS7 Boards	CPM8 driver goes down when under a period of prolonged congestion.
Resolved	IPY00030693	31793	--	SS7 Boards	CPM8 fails with the DM3 board when using heavy media functions, typically around 60 calls per second.
Resolved	IPY00030692	32373	--	SS7 Boards	When setting the address of a calling number in outgoing GSM, the setting is ignored. The destination switch cannot accept the calling number and causes a connection failure.

Issues Sorted By Type, System Release 6.0 cPCI Feature Pack 1 for Windows (Continued)

Issue Type	Defect No.	PTR No.	SU No.	Product or Component	Description
Resolved	IPY00010557	33653	--	SS7 Boards	In a single-SIU configuration during an SIU down-up transition, GCSS7 does not maintain block circuits that were configured to be used by GCSS7 but were not opened by any application at the time of SIU down-up transition.
Resolved	IPY00038708	--	82	Standard Runtime Library (SRL)	An access violation occurs when application calls **sr_waitevtEx( )** for the same device on multiple threads.
Resolved	IPY00032363	--	57	Voice	Random segmentation faults happen due to reading unitialized memory.

[tonegen] section

```
[tonegen]

!!! Initialize the tonegen component.
TgenInit

!!! Delete the default DTMF tone definitions.
DeleteTone 58977
DeleteTone 58978
DeleteTone 58979
DeleteTone 58980
DeleteTone 58981
DeleteTone 58982
DeleteTone 58983
DeleteTone 58984
DeleteTone 58985
DeleteTone 58986
DeleteTone 58987
DeleteTone 58988
DeleteTone 58989
DeleteTone 58990
DeleteTone 58991
DeleteTone 58992

!!! Common Data for All the DTMF definitions.
ToneDesc SegCount 1
ToneDesc SegSigType 1 2
ToneDesc SegAmp1 1 -24
ToneDesc SegAmp2 1 -24
ToneDesc SegReps 1 1
ToneDesc NextSeg 1 65535

!!! DURATIONS
ToneDesc SegOnDur 1 800 !! Currently set to 100ms... (100x8) = 800 samples
ToneDesc SegOffDur 1 800 !! Currently set to 100ms... (100x8) = 800 samples

!!! DTMF 1 ....
ToneDesc SignalId 58977
ToneDesc Label 1
ToneDesc SegFreq1 1 697
ToneDesc SegFreq2 1 1209
CreateTone

!!! DTMF 2
ToneDesc SignalId 58978
ToneDesc Label 2
ToneDesc SegFreq1 1 697
ToneDesc SegFreq2 1 1336
CreateTone

!!! DTMF 3
ToneDesc SignalId 58979
ToneDesc Label 3
ToneDesc SegCount 1
ToneDesc SegFreq1 1 697
ToneDesc SegFreq2 1 1477
CreateTone

!!! DTMF 4
ToneDesc SignalId 58980
ToneDesc Label 4
ToneDesc SegFreq1 1 770
ToneDesc SegFreq2 1 1209
CreateTone
```

```
!!! DTMF 5
ToneDesc SignalId 58981
ToneDesc Label 5
ToneDesc SegFreq1 1 770
ToneDesc SegFreq2 1 1336
CreateTone

!!! DTMF 6
ToneDesc SignalId 58982
ToneDesc Label 6
ToneDesc SegFreq1 1 770
ToneDesc SegFreq2 1 1477
CreateTone

!!! DTMF 7
ToneDesc SignalId 58983
ToneDesc Label 7
ToneDesc SegFreq1 1 852
ToneDesc SegFreq2 1 1209
CreateTone

!!! DTMF 8
ToneDesc SignalId 58984
ToneDesc Label 8
ToneDesc SegFreq1 1 852
ToneDesc SegFreq2 1 1336
CreateTone

!!! DTMF 9
ToneDesc SignalId 58985
ToneDesc Label 9
ToneDesc SegFreq1 1 852
ToneDesc SegFreq2 1 1477
CreateTone

!!! DTMF 0
ToneDesc SignalId 58986
ToneDesc Label 0
ToneDesc SegFreq1 1 941
ToneDesc SegFreq2 1 1336
CreateTone

!!! DTMF a
ToneDesc SignalId 58987
ToneDesc Label a
ToneDesc SegFreq1 1 697
ToneDesc SegFreq2 1 1633
CreateTone

!!! DTMF b
ToneDesc SignalId 58988
ToneDesc Label b
ToneDesc SegFreq1 1 770
ToneDesc SegFreq2 1 1633
CreateTone

!!! DTMF c
ToneDesc SignalId 58989
ToneDesc Label c
ToneDesc SegFreq1 1 852
ToneDesc SegFreq2 1 1633
CreateTone

!!! DTMF d
ToneDesc SignalId 58990
ToneDesc Label d
ToneDesc SegFreq1 1 941
```

```
ToneDesc SegFreq2 1 1633
CreateTone

!!! DTMF #
ToneDesc SignalId 58991
ToneDesc Label #
ToneDesc SegFreq1 1 941
ToneDesc SegFreq2 1 1477
CreateTone

!!! DTMF *
ToneDesc SignalId 58992
ToneDesc Label *
ToneDesc SegFreq1 1 941
ToneDesc SegFreq2 1 1209
CreateTone
```

Documentation Updates

The documentation updates are divided into the following sections, which correspond to the top level categories used on the online documentation navigation page for Intel® Dialogic® System Release 6.0 CompactPCI* Feature Pack 1 for Windows*:

3.1 System Release Documentation

This section contains updates to the following documents (click the title to jump to the corresponding section):

- Intel Dialogic System Release 6.0 cPCI Feature Pack 1 for Windows Release Guide

3.1.1 Intel Dialogic System Release 6.0 cPCI Feature Pack 1 for Windows Release Guide

Update to the **Release Overview** chapter

The following information should replace the existing Redundant System Swap (RSS)/Redundant Host (RH) and Peripheral Hot Swap (PHS) content in the Release Highlights section:

Redundant Host (RH) and Redundant System Slot (RSS)

Redundant Host (RH) technology enables redundant Single Board Computer (SBC) support on cPCI systems. One of the SBCs operates in Active mode while the second (redundant) SBC operates in Standby mode. RH generates a TAKEOVER event when it detects a system-critical fault on the Active SBC.

RH is a standards-based implementation compatible with PICMG 2.0, 2.9, 2.12, 2.16. RH also supports Hot Replace, which allows swapping the host with another compatible SBC without rebooting the operating system. RSS is compatible with PICMG 2.0.

Use of the RH and RSS capabilities requires installation of the Redundant Host Software when you install the system release (or Service Update). The Redundant Host Software is a separate setup

package on the system release navigation screen. (RSS support is part of the Redundant Host Software package.)

Redundant Host support has been implemented on configurations with the following system software and hardware:

- Windows 2000 SP4 only
- MPCHC5085/ZT5524A-1A (Dual CPU) chassis and single board computer
- MPCHC5085/ZT5524A-1B (Single CPU) chassis and single board computer
- ZT5084/ZT5550C chassis and single board computer - Requires Service Update 39 (or later) as well as installation of the Redundant Host Software provided with the Service Update. Also supports RSS.

Peripheral Hot Swap (PHS)

Feature Pack 1 and the Service Update provide Peripheral Hot Swap support for CompactPCI chassis.

Use of the PHS capabilities requires installation of the Hot Swap Kit when you install the system release (or Service Update). The Hot Swap Kit software is a separate setup package on the system release navigation screen.

Following is a complete list of chassis/Single Board Computers on which the swap notification operations are implemented:

Intel NetStructure®	Advantech*	Motorola*	Westek*
MPCHC5085/ ZT5524A-1A MPCHC5085/ ZT5524A-1B ZT5084/ZT5550C† ZT5087/ZT5503† MPCHC5091/ ZT5524A-1A† MPCHC5091/ ZT5524A-1B†	MIC3038/MIC3358 MIC3041/MIC3389† MIC3081/MIC3369†	CPX8216/CPV5350v	P5100/ Advantech MIC-3358
†Requires Service Update 18 (or later) as well as installation of the Hot Swap Kit provided with the Service Update.			

Note: On the Intel NetStructure® configurations, PHS is supported on Windows 2000 SP4 only.

Update to the **Release Overview** chapter

The following information should be added to the IPT1200C description in the Release Highlights section:

- Up to 20% (24) of the IP channels may be used for T.38 relay.

Update to the **Release Overview** chapter

The following product revisions of the Intel NetStructure Combined Media Boards are new in Intel Dialogic System Release 6.0 CompactPCI Feature Pack 1 for Windows and should be included in the Release Highlights section:

- DM/V480A-2T1-CPCI Rev 2 (DMV480A2T1CR2)

- DM/V600A-2E1-CPCI Rev 2 (DMV600A2E1CR2)
- DM/V960A-4T1-CPCI Rev 2 (DMV960A4T1CR2)
- DM/V1200A-4E1-CPCI Rev 2 (DMV1200A4E1CR2)

Updates to the **System Requirements** chapter

Windows Server 2003 Service Pack 1 (SP1), Windows Server 2003 R2, and Windows 2000 Update Rollup 1 for SP4 should be listed as supported operating systems for the Service Update.

The following note should be added to the **Basic Software Requirements** section (PTR# 36031):

Note: Terminal Services Application Server Mode and Active Directory Application Server Mode are not supported on any operating systems.

Updates to the **New Features By Product** chapter

The following information should be removed from the New Features section under New Features for Intel NetStructure DM/IP Series Products:

- T.38 Gateway support

The following information should be removed from the New Features section under New Features for Intel NetStructure IPT Series Products:

- Ability to start RTP stream independently of Global Call using the IP Media Library; provides a mechanism for starting an RTP session when using a 3rd party stack.

Update to the **Supported Hardware** chapter

The following Intel NetStructure Voice with Speech and Conference Revision 2 boards should be documented as follows:

- DM/V480A-2T1-CPCI Rev2 instead of DMV480A2T1CR2
- DM/V600A-2E1-CPCI Rev2 instead of DMV600A2E1CR2
- DM/V960A-4T1-CPCI Rev 2 instead of DMV960A4T1CR2
- DM/V1200A-4E1-CPCI Rev 2 instead of DMV1200A4E1CR2

Update to the **Separately Orderable Products** chapter

The Separately Orderable Products chapter, which refers to the Global Call Protocols Package, is no longer applicable. With the Service Update, the Global Call Protocols Package must now be installed as part of System Release 6.0 CompactPCI Feature Pack 1 for Windows.

3.2 Installation and Configuration Documentation

This section contains updates to the following documents (click the title to jump to the corresponding section):

- Intel Dialogic System Release 6.0 cPCI Feature Pack 1 for Windows Software Installation Guide
- Intel NetStructure for DM3 Architecture for cPCI on Windows Configuration Guide
- Intel NetStructure IPT Series on Windows Configuration Guide

- Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide

3.2.1 Intel Dialogic System Release 6.0 cPCI Feature Pack 1 for Windows Software Installation Guide

Update to **Section 2.3, Performing a Full Installation of the Software**

In Step 8, CORBA is no longer used. See Section 1.6, “New OAMIPC Mechanism Replaces CORBA”, on page 29 of this Release Update.

3.2.2 Intel NetStructure for DM3 Architecture for cPCI on Windows Configuration Guide

Update to **Section 2.6, CT Bus Clock Fallback**

Reference master fallback is not supported and should be deleted from the Section 2.6 introduction and from Figure 5, Clock Fallback. The entire **Section 2.6.2, Reference Master Fallback**, should be deleted. (PTR# 35769)

Update to **Section 3.4, [NFAS] Section**

The third note about NFAS D channel backup (DCBU) supported only on ISDN NI-2 protocol is incorrect. DCBU is supported on 4ESS, 5ESS, and NI-2.

Update to **Chapter 4, Configuration Procedures**

Because of an enhancement in the Service Update, it is no longer necessary to use the fcdgen utility to generate an updated FCD file. All references to fcdgen throughout the Configuration Guide can be ignored. For information on the enhanced way to generate an updated FCD file, see Section 1.5, “Automatic FCD File Generation”, on page 28 of this Release Update.

Update for **Configuring Trunks for Clear Channel Signaling**

Because of a feature introduced in the Service Update, you can now mix ISDN trunks and clear channel trunks on the same DMN160TEC or DMT160TEC board using the Trunk Configuration property sheet of the configuration manager (DCM). Previously, you had to edit the CONFIG file to accomplish this. The information in **Section 3.8, Configuring Trunks for Clear Channel Signaling**, is no longer applicable, and the information in **Section 5.8, Trunk Configuration Property Sheet**, should be updated for this new feature. For information about the changes to the Trunk Configuration property sheet, see Section 1.19, “Mixing ISDN and Clear Channel on a DMN160TEC or DMT160TEC Board”, on page 51 of this Release Update.

Update for **Dynamic Selection of Signaling Type for a Trunk**

Because of a feature introduced in the Service Update, you can now change the signaling type at runtime of any trunk from ISDN to clear channel or vice versa on a DMN160TEC or DMT160TEC board for E1 configurations only. In order to enable this capability, a new parameter, **Dynamic_ISDN_CC**, has been added to the Trunk Configuration property sheet, so the information in **Section 5.8, Trunk Configuration Property Sheet**, should be updated for this new parameter. For information about this feature and the new parameter, see Section 1.10, “Dynamic Selection of Signaling Type for a Trunk”, on page 33 of this Release Update.

Updates to **Section 5.5, Physical Property Sheet**

The description of the DCM parameter **PhysicalSlotNumber** should be replaced by the following:

PhysicalSlotNumber (CompactPCI Boards)

Description: The **PhysicalSlotNumber** parameter specifies the number of the physical slot in which the CompactPCI board is installed.

Values: A positive integer or hexadecimal value.

Guidelines: The **PhysicalSlotNumber** parameter is read-only and only applies to CompactPCI boards. A value of 1 indicates the first slot in the chassis. (The chassis slot numbers are usually marked on the front of the chassis.)

The description of the DCM parameter **PciID** should be replaced by the following:

PciID

Description: The **PciID** parameter is a positive integer or hexadecimal value in which the lower 5 bits specify a board's rotary-switch setting (PCI boards) or the physical slot number location of the board (CompactPCI boards). The rotary-switch setting for PCI boards can be the same for all boards in the system if the value is set to 0.

Values: A positive integer or hexadecimal value

Guidelines: The **PciID** parameter is set by the system software and should not be changed by the user.

Update to **Section 6.7, [0x3b] Parameters**

Because of a feature introduced in the Service Update, three new parameters should be documented in Section 6.7, [0x3b] Parameters. The parameters are **CSUMS_AGC_k_Def** (AGC K Constant), **CSUMS_AGC_low_threshold** (AGC Noise Level Lower Threshold), and **CSUMS_AGC_max_gain** (AGC Maximum Gain.) For information about these parameters, see Section 1.25, “Enhanced Volume Control for Conferencing”, on page 66 of this Release Update.

Update to **Section 6.9, [lineAdmin.x] Parameters (Digital Voice)**

In the guidelines for the **SignalingType** parameter, the note about NFAS D channel backup (DCBU) supported only on ISDN NI-2 protocol is incorrect. DCBU is supported on 4ESS, 5ESS, and NI-2.

Update to **Section 6.12, [NFAS.x] Parameters**

In the description of the **NFAS_Standby_IntID** parameter, the note about NFAS D channel backup (DCBU) supported only on ISDN NI-2 protocol is incorrect. DCBU is supported on 4ESS, 5ESS, and NI-2.

Update to **Section 6.18, [CCS] Parameters**

Because of a feature introduced in the Service Update, ISDN network-side mode is now supported and fully qualified for operation in a deployment environment. In Section 6.18, [CCS] Parameters, the description of the **CCS_PROTOCOL_MODE**

parameter should be updated to indicate this. The following note is no longer applicable:

Note: With the exception of the Q-SIG protocol (where the User-side and Network-side protocols are symmetrical), using the CCS_PROTOCOL_MODE parameter to configure a Network-side protocol is supported for back-to-back testing purposes only. The Network-side firmware is not fully qualified for operation in a deployment environment.

For further information about this feature, see Section 1.13, "ISDN Network Side Conformance to Network Protocol Standards ITU-T Q921 and Q931", on page 42 of this Release Update.

3.2.3 Intel NetStructure IPT Series on Windows Configuration Guide

There are currently no updates to this document.

3.2.4 Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide

Update to **Section 2.4.2, Downloading the Protocol and CDP File on a Windows System**

The following note should be added after the first paragraph of this section (PTR# 36373):

Note: If the pdk.cfg file is not present in the %INTEL_DIALOGIC_CFG% directory and pdkmanagerregsetup is run, no indication is given that a problem exists. Subsequent attempts to start the Intel Dialogic services will fail with no discernible error.

3.3 OA&M Documentation

This section contains updates to the following documents (click the title to jump to the corresponding section):

- Intel Dialogic System Release 6.0 on cPCI for Windows Administration Guide
- Diagnostics Guide
- SNMP Agent Software for Windows Administration Guide
- Third Party Hardware TDM Bus Administration for Windows
- Board Management API Library Reference
- Event Service API Programming Guide
- Event Service API Library Reference
- Native Configuration Management API Programming Guide
- Native Configuration Management API Library Reference

3.3.1 Intel Dialogic System Release 6.0 on cPCI for Windows Administration Guide

Update to Chapter 2, **Stopping and Starting the System**

Information about the **Start Server Only** mode has been added to the Administrative Guide for FP1as follows:

Select the **Start Server Only** mode from the **System/Device autostart** submenu in the DCM. This causes the Intel Dialogic System to start automatically when the server is started. The boards will be automatically detected, but not started.

3.3.2 Diagnostics Guide

Update to **Chapter 4, Checking DM3 Architecture Boards**

The following section should be added:

Diagnosing Communication Link Errors between HDSI Board and SIB

Communication failures between the HDSI board and the Station Interface Box (SIB) will cause all active calls on that link to be terminated. Each cable connected between the HDSI board and the SIB represents one communication link and can carry data and signaling for up to 30 channels. A communication error between the HDSI board and the SIB generates an error like the following in the DebugAngel.log file:

```
14:53:22.086| 003:CP1:ESI : Msg ACK timeout, line 0, timer 2, sent 20227302, curr 20227802
14:53:23.118| 003:CP1:Line 0 : Warning - ESI message 0x2 read in CONFIG state
```

This message means that a cable check response event didn't occur in the specified time. This will cause **all** channels on the associated link to be terminated. This will last only a short period of time (2 seconds or less) and will disappear once the HDSI board is able to communicate reliably with the SIB. Once this condition disappears, call processing can continue.

When calls are terminated, there is no asynchronous event that will be sent to the application.

Note: The same type of "alarm" behavior will occur if the cable is unplugged or the SIB is powered down.

Workaround: To handle this alarm condition, update the application to check the current hook state of the channel prior to starting any operations. Checking of the hook state can be done using the MSI API call **ATMS_TSSGBIT( )**. If the hook state was MS_ONHOOK and it was expected to be MS_OFFHOOK, the application could then take the appropriate action based on the hook state.

Update to **Chapter 28, Runtime Trace Facility (RTF) Reference** (IPY00037518)

The following information about using binary log files should be added to **Section 28.3.2, Logfile Tag**:

For installations with high channel densities, or which have enabled all or most RTF trace levels, the volume of logging may result in an increased CPU utilization by the RtfServer executable as a result of the increased volume of log messages.

As shipped, the RTF log files are generated in ASCII text mode. There is a configuration parameter in the RTF configuration file (*RtfConfigWin.xml* for Windows, *RtfConfigLinux.xml* for Linux) that allows log files to be generated in either "text" or

"binary" format. Testing on high channel density systems with most or all of the RTF trace levels enabled has shown that the generation of binary format RTF log files has less of an impact on CPU usage than does the generation of text format RTF log files.

If the volume of logging results in high CPU usage, then using binary format will reduce the usage.

Enabling Binary Format RTF Log Files

The XML file contains the following line, which allows changes to log file parameters to be made:

```
<Logfile path="$(INTEL_DIALOGIC_DIR)\log" size="300" maxbackups="10"
preserve_size="300" preserve_maxbackups="10"
duplicate_to_debug_console="0" log_format="text" />
```

The "log_format" value controls the type of log files that are written. Valid values for this parameter are "text" and "binary". Once a change has been made to the XML file, it must be reloaded using the rtftool reload command.

Converting Binary Format RTF Log Files to Text Format

In order for binary log files to be examined, they must be converted into text format. This can be done by using the rtftool export command.

```
rtftool export [-d source_dir | -s source_file]
[-f [dest_file] | -m dest_dir]
```

By default, the name of the text format files generated by this command will be *EXPORT-<RTF binary log file name>*. For example, if the binary format file is named *rtflog-LOCAL-20070306-15h09m26.506s.txt*, then the default name of the generated text format file will be *EXPORT-rtflog-LOCAL-20070306-15h09m26.506s.txt.* This behavior can be overridden using the -f command line option.

The rtftool utility is a stand-alone program, and it is not necessary to have the Dialogic System Release installed on the system in order to convert RTF log files from binary to text format.

Note: When generating large binary files with RTF, do not split the single large binary file and then use the individual split files with the rtftool utility. Rtftool will not work with chopped binary files.

Update to **Chapter 29, RTFManager Reference** (IPY00037518)

Section 29.5, General Tab, says that binary log format is not supported by the current release. This is not correct; binary log format is supported. For information about binary log files, see the update to **Chapter 28, Runtime Trace Facility (RTF) Reference** above.

3.3.3 SNMP Agent Software for Windows Administration Guide

There are currently no updates to this document.

3.3.4 Third Party Hardware TDM Bus Administration for Windows

There are currently no updates to this document.

3.3.5 Board Management API Library Reference

There are currently no updates to this document.

3.3.6 Event Service API Programming Guide

There are currently no updates to this document.

3.3.7 Event Service API Library Reference

Update to the **Data Structures** chapter

In the **NetworkEventMsg** section, the description of "auid" should be changed to the following (PTR# 32157):

auid

AUID of the dtiB<n> virtual board that generated the network alarm (n starting with 1).

The network alarms are against Digital Telephony Interface (DTI) interfaces. Therefore, the event that is broadcast concerns a dtiB<n> virtual board, and the event contains the AUID of the virtual board. Note that Intel NetStructure® DM/T and DM/V boards can have up to 16 network interfaces, so <n> can be an integer between 1 and 16 (dtiB<n>). Clock Fallback concerns the ability of the board to be Clock Master, so this event is sent with the AUID of the physical board.

3.3.8 Native Configuration Management API Programming Guide

There are currently no updates to this document.

3.3.9 Native Configuration Management API Library Reference

Updates to **NCM_ApplyTrunkConfiguration( )**

On the **NCM_ApplyTrunkConfiguration( )** function reference page, the first two inputs for **NCM_ApplyTrunkConfiguration( )** are not pointers. Therefore the words "pointer to a" should be deleted from the descriptions of the inputs **NCMFamily* pncmFamily** and **NCMDevice* pncmDeviceUnique**. (PTR# 36260)

On the **NCM_ApplyTrunkConfiguration( )** function reference page, the following two parameters should be added:

pMediaLoad	pre-defined sets of supported features. A media load consists of a configuration file set and associated firmware. Universal media loads support voice, fax, and conferencing resources simultaneously. See the "Media Loads Supported" table below.
pErrorMsg	API provides as needed. This is a pointer to BYTE that the API fills with an error message on return. BYTE is defined as an unsigned character.

Media Loads Supported

Boards	Media Loads	Protocols
DMV1200BTEP	UL1	Group 1: CAS, 4ESS, 5ESS, DMS, NI2, NTT, QSIGT1, QSIGE1, NET5, R2MF, T1CC, E1CC
		Group 2: DPNSS, DASS2
	UL2	Group 1: CAS, 4ESS, 5ESS, DMS, NI2, NTT, QSIGT1, QSIGE1, NET5, R2MF, T1CC, E1CC
		Group 2: DPNSS, DASS2
	UL3	Group 1: CAS, 4ESS, 5ESS, DMS, NI2, NTT, QSIGT1, QSIGE1, NET5, R2MF, T1CC, E1CC
		Group 2: DPNSS, DASS2
	ML10B	Group 1: CAS, 4ESS, 5ESS, DMS, NI2, NTT, QSIGT1, QSIGE1, NET5, R2MF, T1CC, E1CC
		Group 2: DPNSS, DASS2
	ML10	Group 1: CAS, 4ESS, 5ESS, DMS, NI2, NTT, QSIGT1, QSIGE1, NET5, R2MF, T1CC, E1CC
		Group 2: DPNSS, DASS2
DMV600BTEP	UL1	Group 1: CAS, 4ESS, 5ESS, DMS, NI2, NTT, QSIGT1, QSIGE1, NET5, R2MF, T1CC, E1CC
		Group 2: DPNSS, DASS2
	UL2	Group 1: CAS, 4ESS, 5ESS, DMS, NI2, NTT, QSIGT1, QSIGE1, NET5, R2MF, T1CC, E1CC
		Group 2: DPNSS, DASS2
DMT160TEC	Tone	Group 1: 4ESS, 5ESS, DMS, NI2, NTT, QSIGT1, QSIGE1, NET5, ISDNT1CC, ISDNE1CC
		Group 2: CAS, T1CC
		Group 3: R2MF, E1CC
DMN160TEC	Network Only	Group 1: 4ESS, 5ESS, DMS, NI2, NTT, QSIGT1, QSIGE1, NET5, ISDNT1CC, ISDNE1CC
	ML10	Group 1: CAS, 4ESS, 5ESS, DMS, NI2, NTT, QSIGT1, QSIGE1, NET5, R2MF, T1CC, E1CC
		Group 2: DPNSS, DASS2
	UL2	Group 1: CAS, 4ESS, 5ESS, DMS, NI2, NTT, QSIGT1, QSIGE1, NET5, R2MF, T1CC, E1CC
		Group 2: DPNSS, DASS2

Boards	Media Loads	Protocols
	ML10B	Group 1: CAS, 4ESS, 5ESS, DMS, NI2, NTT, QSIGT1, QSIGE1, NET5, R2MF, T1CC, E1CC
		Group 2: DPNSS, DASS2
DMV600BTEC	UL1	Group 1: CAS, 4ESS, 5ESS, DMS, NI2, NTT, QSIGT1, QSIGE1, NET5, R2MF, T1CC, E1CC
		Group 2: DPNSS, DASS2

On the **NCM_ApplyTrunkConfiguration( )** function reference page, the sample code should be updated to show the two new parameters:

```
#include <stdio.h>
#include "ncmapi.h"
void main()
{
NCMRetCode ncmRetCode;
char buffer[300] = {0};

         NCMFamily family;
         family.name = "DM3";
         family.next = NULL;

         NCMDevice UniqueName;
         UniqueName.name = "DMV1200BTEP #1 in slot 2/10";
         UniqueName.next = NULL;

         NCMTrunkConfig ncmTruckConfig[4] = {0};

         NCMFeatureType ncmFeatureType = {0};

         ncmTruckConfig[0].TrunkName  = "Trunk1";
         ncmTruckConfig[0].TrunkValue = "4ESS(T1, Group 1)";
         ncmTruckConfig[0].next      = &(ncmTruckConfig[1]);

         ncmTruckConfig[1].TrunkName  = "Trunk2";
         ncmTruckConfig[1].TrunkValue = "4ESS(T1, Group 1)";
         ncmTruckConfig[1].next      = &(ncmTruckConfig[2]);

         ncmTruckConfig[2].TrunkName  = "Trunk3";
         ncmTruckConfig[2].TrunkValue = "5ESS(T1, Group 1)";
         ncmTruckConfig[2].next      = &(ncmTruckConfig[3]);

ncmTruckConfig[3].TrunkName  = "Trunk4";
         ncmTruckConfig[3].TrunkValue = "4ESS(T1, Group 1)";
         ncmTruckConfig[3].next = NULL;

         strncpy(ncmFeatureType.MediaLoad, "ML10", MEDIA_LOAD_LENGTH);

ncmRetCode = NCM_ApplyTrunkConfiguration(family,UniqueName,
ncmTruckConfig, &ncmFeatureType, reinterpret_cast<unsigned
```

```
char*>(buffer));
         if (ncmRetCode != NCM_SUCCESS)
         {
           printf("Error calling NCM_ApplyTrunkConfiguration(). It
returned: %d \n", ncmRetCode;
           printf( " Error Msg:  %s \n", buffer);
         }
              else
              {
                    printf("SUccessful calling
NCM_ApplyTrunkConfiguration\n");
              }
              printf("press any key to exit\n");
              getchar();
}
...
```

Two new functions

The following two functions should be added to this document:

- **NCM_RestoreConfiguration( )**
- **NCM_SaveConfiguration(**)

The function reference information is provided below.

NCM_RestoreConfiguration( )

Name: NCMRetCode NCM_RestoreConfiguration(dwWaitTime)

Inputs: int dwWaitTime
- number of seconds the system waits before applying the saved configuration

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h
NCMTypes.h

Category: System administration

Mode: synchronous

Description

The **NCM_RestoreConfiguration( )** function allows an application to apply a previously saved system configuration. Invoke the **NCM_SaveConfiguration( )** function to save a system configuration before invoking the **NCM_RestoreConfiguration( )** to restore the saved configuration. This allows the application to update the complete system configuration after a failover occurs. This function should be used only after the failover is completed.

Parameter	Description
dwWaitTime	indicates the number of seconds the system waits before applying the saved configuration. *Note:* This value does not indicate the number of seconds for the function to complete.

Cautions

- This function is currently applicable to cPCI systems with redundant host capabilities.
- This function cannot be called while the Intel Dialogic system service is running.
- Intel NetStructure® IPT Series boards require an additional 90 seconds for reconfiguration after this function is invoked.

Errors

Possible errors for this function include:

ERROR_NO_FREE_RESOURCES
resources are not available

ERROR_NOT_ENOUGH_MEMORY
required memory space is not available

ERROR_OUTOFMEMORY
function call does not have enough memory to execute

MEMORY_ALLOCATION_ERROR
memory has been allocated incorrectly

NCME_BUS_CONFIG_NOT_FOUND
a warning that is issued when the system cannot locate a previously saved system configuration

NCME_CTBB_DEVICESDETECTED
function has already detected CTBB devices

NCME_DATA_NOT_FOUND
data not found

NCME_FAILED_RESTORE
an error occurred while attempting to restore the saved configuration

NCME_GENERAL
general error

NCME_MEM_ALLOC
memory allocation error

NCME_NO_SETUP_FOUND
saved configuration could not be found

NCME_SYSTEM_RUNNING
Intel Dialogic system service is currently running

NCM_NOT_ALL_BOARDS_DETECTED_TIMEOUT
all boards stored in the saved configuration could not be found

NO_INF_FILES_FOUND
.inf files could not be found

Note: In addition to the error codes listed above, this function may also return any of the **NCM_GetInstalledFamilies( )**, **NCM_GetInstalledDevices( )**, **NCM_DetectBoardsEx( )** or **NCM_SetValueEx( )** function error codes.

Example

```
#include <NCMApi.h>

void main(void)
{
   NCMFamily *ncmFamily = NULL;
   NCMDevice *ncmDevice = NULL;
   int dwtimeout = 30;
   NCMRetCode ncmRc = NCME_GENERAL;
   ...
   // Failover occurred and ready to configure the new Active.

   ncmRc = NCM_RestoreConfiguration(dwtimeout);
   if (NCM_SUCCESS != ncmRc)
   {
      // Process Error
      if (NCM_NOT_ALL_BOARDS_DETECTED_TIMEOUT == ncmRc)
      {
```

```
            dwtimeout = 60;
            ncmRc = NCM_RestoreConfiguration(dwtimeout); // Increase the timeout
        }
    }


...// Download the system
...// Do rest of the activity
}
```

See Also

- **NCM_GetDialogicDir( )**
- **NCM_DetectBoardsEx( )**
- **NCM_GetInstalledDevices( )**
- **NCM_GetInstalledFamilies( )**
- **NCM_GetValueEx( )**
- **NCM_SaveConfiguration( )**

NCM_SaveConfiguration( )

Name: NCMRetCode NCM_SaveConfiguration(void)

Returns: NCM_SUCCESS if success
NCM error code if failure

Includes: NCMApi.h
NCMTypes.h

Category: System administration

Mode: synchronous

Description

The **NCM_SaveConfiguration( )** function allows applications to save the current system configuration for future use.

Cautions

- This function is currently applicable to cPCI systems with redundant host capabilities.
- This function should only be called after the system has been completely configured.

Errors

Possible errors for this function include:

NCME_DATA_NOT_FOUND
data not found

NCME_FAILED_SAVE
an error occurred saving the system configuration

NCME_GENERAL
general error

NCME_MEM_ALLOC
memory could not be allocated to perform the function

Note: In addition to the error codes listed above, this function may also return any of the **NCM_GetInstalledFamilies( )**, **NCM_GetInstalledDevices( )** or **NCM_GetValueEx( )** function error codes.

Example

```
#include <NCMApi.h>

void main(void)
{
   NCMFamily *ncmFamily = NULL;
   NCMDevice *ncmDevice = NULL;
   NCMRetCode ncmRc = NCM_SUCCESS;

   ...
```

```
    NCM_DetectBoardsEx(…)
    // Setup the system configurations. This step is needed only once.
    ncmRc = NCM_SaveConfiguration();
    if (NCM_SUCCESS != ncmRc)
       {
          // Process error here, Invoke it again not a harmful function.
          ncmRc = NCM_SaveConfiguration();
       }
        // Do rest of the activity
}
```

See Also

- **NCM_GetDialogicDir( )**
- **NCM_GetInstalledDevices( )**
- **NCM_GetInstalledFamilies( )**
- **NCM_GetValueEx( )**
- **NCM_RestoreConfiguration( )**

3.4 Programming Libraries Documentation

This section contains updates to the following documents (click the title to jump to the corresponding section):

- Audio Conferencing API Programming Guide
- Audio Conferencing API Library Reference
- Continuous Speech Processing API Programming Guide
- Continuous Speech Processing API Library Reference
- Digital Network Interface Software Reference
- Fax Software Reference
- Global Call API Programming Guide
- Global Call API Library Reference
- Global Call E1/T1 CAS/R2 Technology User's Guide
- Global Call IP Technology Guide
- Global Call ISDN Technology Guide
- Global Call SS7 Technology Guide
- IP Media Library API Programming Guide
- IP Media Library API Library Reference
- Modular Station Interface API Programming Guide
- Modular Station Interface API Library Reference
- Porting Global Call H.323 Applications from Embedded Stack to Host-Based Stack Application Note
- Standard Runtime Library API Programming Guide
- Standard Runtime Library API Library Reference
- Voice API Programming Guide
- Voice API Library Reference

3.4.1 Audio Conferencing API Programming Guide

There are currently no updates to this document.

3.4.2 Audio Conferencing API Library Reference

There are currently no updates to this document.

3.4.3 Continuous Speech Processing API Programming Guide

There are currently no updates to this document.

3.4.4 Continuous Speech Processing API Library Reference

There are currently no updates to this document.

3.4.5 Digital Network Interface Software Reference

There are currently no updates to this document.

3.4.6 Fax Software Reference

Update to **Section 6.2.2, Storing Incoming Fax Data** (Defect# IPY00031917 = PTR# 27337)

The following note should be added under **Storing in Multiple TIFF/F Files**:

Note: If **fx_sendfax( )** is called to send a multiple-page TIFF/F with io_phdcont=DFC_EOM, once the first page of the fax is received, a TDX_PHASED event is issued but no TFX_FAXRECV event is returned. TFX_FAXRECV is returned when all fax pages are transmitted.

Update to **DF_IOTT** data structure

The following information should be added to the io_length field description in Table 14, DF_IOTT Fields (PTR# 30192):

When specifying that data be transferred until the end of the file is reached (io_type set to IO_DEV and io_length set to -1 in the DF_IOTT structure), you may encounter compiler warnings. This occurs because io_length is defined as unsigned long. The fax software allows for signed values, so you can ignore any compiler warnings related to this situation.

Update to **Section 10.3.2, DF_ASCIIDATA Field Descriptions** (IPY00037855)

In **Table 12, DF_ASCIIDATA Fields**, the description of the font field states that the default fax font provides 10 characters per inch spacing. This statement should be changed as follows:

The default fax font character spacing may differ on various Windows® operating system releases depending on the OEM font supplied on the particular system.

3.4.7 Global Call API Programming Guide

There are currently no updates to this document.

3.4.8 Global Call API Library Reference

Update to **Global Call Function Support by Technology** section

Because of a new Service Update, this section should be updated to indicate that the GCAMS functions (with the exception of **gc_TransmitAlarms( )** and **gc_StopTransmitAlarms( )**) are supported for SS7 technology. Also, the individual GCAMS function reference pages should be updated to indicate support for SS7 technology.

Update to **gc_DropCall( )**

On the **gc_DropCall( )** function reference page, the following caution should be added (PTR# 34237):

- With CAS protocols, the GCEV_DROPCALL event may be delayed when **gc_DropCall**( ) is called. GCEV_DROPCALL is sent to the application only when the channel becomes Idle. This is expected behavior of a CAS protocol. In the Offered state (ringing), there is no way for the receiving side to tell the calling side to stop ringing. It will not happen until the CO times out (ring no answer) and drops their bits to IDLE. That is why the GCEV_DROPCALL event may seem to be "delayed" in this situation; it is affected by the time-out time governed by CO.

Update to **gc_InitXfer( )** (IPY00038401)

In the code example, the **gc_InitXfer( )** parameters are shown in the wrong order. The line:

```
if (gc_InitXfer(pline->crn, NULL, &gc_pRetParmBlk, EV_ASYNC) == -1)
```

should be changed to:

```
if (gc_InitXfer(pline->crn, &gc_pRetParmBlk, NULL, EV_ASYNC) == -1)
```

Update to **gc_MakeCall( )**

On the **gc_MakeCall( )** function reference page, the following caution should be added (PTR# 33852):

- In synchronous mode, calls to **gc_MakeCall**( ) must be serialized. Multiple gc_MakeCalls cannot be made on the same channel from multiple threads.

Update to **GCLIB_MAKECALL_BLK**

On the GCLIB_MAKECALL_BLK data structure reference page, in the Description section, GC**MAKE**CALLBLK_DEFAULT should be changed to GC**MK**CALLBLK_DEFAULT.

Update for **Tracing CAS Signaling**

Because of enhancements for tracing CAS signaling introduced in the Service Update, the **gc_StartTrace( )**, **gc_StopTrace( )**, and **gc_SetConfigData( )** functions are now supported for DM3 E1/T1 technology. This should be indicated in Table 1, Global Call Function Support by Technology, and in the individual function reference pages in the *Global Call API Library Reference*. In addition, the GCEV_TRACEDATA event should be added to the **Events** chapter, and the GC_TRACEDATA data structure should be added to the **Data Structures** chapter. Detailed information about tracing CAS signaling is now included in the Diagnostics Guide.

Update for **Dynamic Selection of the Signaling Type for a Trunk**

Because of a new feature in the Service Update, a new EGC_NOT_ENABLED error code has been defined that should be documented in the **Error Codes** chapter. For further information about this new feature and the new error code, see Section 1.10, "Dynamic Selection of Signaling Type for a Trunk", on page 33 of this Release Update.

3.4.9 Global Call E1/T1 CAS/R2 Technology User's Guide

Updates to **Chapter 4, Applying Global Call Functions to E-1 CAS or T-1 Robbed Bit Applications**

In **Section 4.13, gc_MakeCall( )**, the information about PDK protocols should be changed as follows (the change is in the third bullet) (PTR# 29448):

For PDK protocols, the time-out value used is determined by:

- The **timeout** parameter in the **gc_MakeCall( )** function.
- The **PSL_DefaultMakeCallTimeout** parameter specified in the .cdp file if the **timeout** parameter in the **gc_MakeCall( )** function is 0 and call analysis is not specified.
- The **PSL_CallProgressMaxDialingTime** parameter specified in the .cdp file if the **timeout** parameter in the **gc_MakeCall( )** function is 0, call analysis is specified, and **PSL_DefaultMakeCallTimeout** is less than **PSL_CallProgressMaxDialingTime**.

Note: PDK protocols do not use the outbound number of ringback tones to define the time-out.

In **Section 4.18, gc_SetChanState( )**, the following note should be added (PTR# 36726):

Note: When a channel is set to out-of-service state, not all protocols send the blocking pattern by default. For such protocols, a parameter in the .cdp file has to be set to the appropriate value so that the blocking pattern is sent when the channel is put out-of-service. Refer to the *Global Call Country Dependent Parameters (CDP) for PDK Protocols Configuration Guide* for more information.

3.4.10 Global Call IP Technology Guide

Updates for retrieving coder information from fast start call offers

Because of a new feature introduced in the Service Update, information about retrieving coder information from fast start call offers should be added to **Chapter 4, IP-Specific Operations**. Related to this feature is a change to the IP_VIRTBOARD data structure. For information about this feature, see Section 1.11, “Retrieving Coder Information from Fast Start Call Offers”, on page 39 of this Release Update.

Updates for H.323 Annex M tunneled signaling messages

Because of a new feature introduced in the Service Update, information about using H.323 Annex M tunneled signaling messages should be added to **Chapter 4, IP-Specific Operations**. Related to this feature are a new parameter set, IPSET_TUNNELEDSIGNALMSG, a new data structure, IP_TUNNELPROTOCOL_ALTID, as well as a change to the IP_VIRTBOARD data structure. For information about this feature, see Section 1.20, “Using H.323 Annex M Tunneled Signaling Messages”, on page 54 of this Release Update.

Update to the **Call Control Library Initialization** section

The following new subsection is added under **Section 4.1, Call Control Library Initialization** (page 60):

4.1.1 Setting a SIP Outbound Proxy

When initializing a board device for use with SIP, the application can set an outbound proxy. When such a proxy is set, all outbound requests are sent to the proxy address rather than the IP address of the original RequestURI. The proxy is set by specifying an IP address or a hostname in the IP_VIRTBOARD structure that is used in the **gc_Start( )** function; if both an IP address and a hostname are specified, the IP address will take precedence.

The following code snippet illustrates how to set a SIP outbound proxy for a single board:

```
#include "gclib.h"
..
..
#define BOARDS_NUM 1
..
..
/* initialize start parameters */
IPCCLIB_START_DATA cclibStartData;
memset(&cclibStartData,0,sizeof(IPCCLIB_START_DATA));
IP_VIRTBOARD virtBoards[BOARDS_NUM];
memset(virtBoards,0,sizeof(IP_VIRTBOARD)*BOARDS_NUM);

/* initialize start data */
INIT_IPCCLIB_START_DATA(&cclibStartData, BOARDS_NUM, virtBoards);

/* initialize virtual board */
INIT_IP_VIRTBOARD(&virtBoards[0]);

// set outbound proxy by IP Address
virtBoards[0].outbound_proxy_IP.ip_ver = IPVER4;
virtBoards[0].outbound_proxy_IP.u_ipaddr.ipv4 = inet_addr("192.168.1.227");

// set outbound proxy by hostname.
// if outbound proxy is also set by IP address, this is ignored
char OutboundProxyHostName[256];
strcpy(OutboundProxyHostName,"my_outbound_proxy");
virtBoard[0].outbound_proxy_hostname = OutboundProxyHostName;

// set outbound proxy port
virtBoards[0].outbound_proxy_port = 5060;
```

Update to the **Specifying Nonstandard Data Information When Using H.323** section **Section 4.3.3, Specifying Nonstandard Data Information When Using H.323** (page 69) is replaced with the following information to describe support for Nonstandard Data that is longer than 255 bytes:

4.3.3 Specifying Nonstandard Data Information When Using H.323

To specify Nonstandard Data information to be included in the H.323 SETUP message, use the **gc_SetUserInfo( )** function with a **duration** parameter of GC_SINGLECALL to preset the information. If the **duration** parameter is set to GC_ALLCALLS, the function fails.

To specify Nonstandard Data, the GC_PARM_BLK pointed by the **infoparmblkp** parameter in the function call must be contain two parameter elements that use the IPSET_NONSTANDARDDATA parameter set ID. The first required parameter element specifies the Nonstandard Data itself, and the second parameter element identifies the type of object identifier to use.

The maximum length of the Global Call parameter used for the Nonstandard Data information is configured at start-up via the max_parm_data_size field in the IPCCLIB_START_DATA structure. The default size is 255 (for backwards compatibility), but applications may configure it to be as large as 4096 bytes. Applications must use the extended **gc_util_..._ex( )** functions to insert or extract any GC_PARM_BLK parameter elements whose data length is defined to be greater than 255.

Note: In practice, applications may not be able to utilize the full maximum length of the nonstandard data parameter element as configured in max_parm_data_size. The H.323 stack limits the overall size of messages to be max_parm_data_size + 512 bytes, and any messages that exceed this limit are truncated without any notification to the application.

The parameter element for the Nonstandard Data data is:

IPSET_NONSTANDARDDATA
    IPPARM_NONSTANDARDDATA_DATA
    - value = Nonstandard Data string, max length = max_parm_data_size (configurable at library start-up)

The parameter element for the Nonstandard Data identifier is one (and only one) of the following:

IPSET_NONSTANDARDDATA
    IPPARM_NONSTANDARDDATA_OBJID
    - value = array of unsigned integers, max length = MAX_NS_PARM_OBJID_LENGTH

IPSET_NONSTANDARDDATA
    IPPARM_H221NONSTANDARD
    - value = IP_H221NONSTANDARD structure

The following code example shown how to set nonstandard data elements:

```
IP_H221NONSTANDARD appH221NonStd;
appH221NonStd.country_code = 181;
appH221NonStd.extension = 31;
appH221NonStd.manufacturer_code = 11;
char* pData = "Data String";
char* pOid = "1 22 333 4444";
choiceOfNSData = 1;/* App decides which type of object identifier to use */

/* setting NS Data */
gc_util_insert_parm_ref_ex(&pParmBlock,
                           IPSET_NONSTANDARDDATA,
                           IPPARM_NONSTANDARDDATA_DATA,
                           (unsigned long)(strlen(pData)+1),
                           pData);

   if (choiceOfNSData) /* App decides the CHOICE of OBJECTIDENTIFIER.
                          It cannot set both objid & H221 */
   {
      gc_util_insert_parm_ref(&pParmBlock,
                              IPSET_NONSTANDARDDATA,
                              IPPARM_H221NONSTANDARD,
                              (unsigned char)sizeof(IP_H221NONSTANDARD),
                              &appH221NonStd);
   }

   else
   {
      gc_util_insert_parm_ref(&pParmBlock,
                              IPSET_NONSTANDARDDATA,
                              IPPARM_NONSTANDARDDATA_OBJID,
                              (unsigned char)(strlen(pOid)+1),
                              pOid);
   }
```

Update to the **Specifying Nonstandard Control Information When Using H.323** section

Section 4.3.4, Specifying Nonstandard Control Information When Using H.323 (pages 69-70) is replaced by the following information to reflect support for Nonstandard Control information that is longer than 255 bytes:

4.3.4 Specifying Nonstandard Control Information When Using H.323

To specify Nonstandard Control information to be included in the H.323 SETUP message, use the **gc_SetUserInfo( )** function with a **duration** parameter of GC_SINGLECALL to preset the information. If the **duration** parameter is set to GC_ALLCALLS, the function fails.

To specify Nonstandard Control data, the GC_PARM_BLK pointed by the **infoparmblkp** function must be set up with two parameter elements that use the IPSET_NONSTANDARDCONTROL parameter set ID. The first required parameter element specifies the Nonstandard Control data itself, and the second parameter element identifies the type of object identifier to use.

The maximum length of the Global Call parameter used for the Nonstandard Control information is configured at start-up via the max_parm_data_size field in the IPCCLIB_START_DATA structure. The default size is 255 (for backwards compatibility), but applications may configure it to be as large as 4096 bytes. Applications must use the extended **gc_util_..._ex( )** functions to insert or extract any GC_PARM_BLK parameter elements whose data length is defined to be greater than 255.

Note: In practice, applications may not be able to utilize the full maximum length of the nonstandard control parameter element as configured in max_parm_data_size. The H.323 stack limits the overall size of messages to be max_parm_data_size + 512 bytes, and any messages that exceed this limit are truncated without any notification to the application.

The parameter element for the Nonstandard Control data is:

IPSET_NONSTANDARDCONTROL
IPPARM_NONSTANDARDDATA_DATA
- value = Nonstandard Data string, max length = max_parm_data_size (configurable at library start-up)

The parameter element for the Nonstandard Control identifier is one (and only one) of the following:

IPSET_NONSTANDARDCONTROL
IPPARM_NONSTANDARDDATA_OBJID
- value = array of unsigned integers, max length = MAX_NS_PARM_OBJID_LENGTH

IPSET_NONSTANDARDCONTROL
IPPARM_H221NONSTANDARD
- value = IP_H221NONSTANDARD structure

The following code example shows how to set nonstandard data elements:

```
IP_H221NONSTANDARD appH221NonStd;
appH221NonStd.country_code = 181;
appH221NonStd.extension = 31;
appH221NonStd.manufacturer_code = 11;
char* pControl = "Control String";
char* pOid = "1 22 333 4444";
choiceOfNSControl = 1; /* App decides which type of object identifier to use */

/* setting NS Control */
gc_util_insert_parm_ref_ex(&pParmBlock,
                           IPSET_NONSTANDARDCONTROL,
                           IPPARM_NONSTANDARDDATA_DATA,
                           (unsignned long)(strlen(pControl)+1),
                           pControl);

   if (choiceOfNSControl)  /* App decide the CHOICE of OBJECTIDENTIFIER.
                              It cannot set both objid & h221 */
   {
      gc_util_insert_parm_ref(&pParmBlock,
                              IPSET_NONSTANDARDCONTROL,
                              IPPARM_H221NONSTANDARD,
                              (unsingned char)sizeof(IP_H221NONSTANDARD),
                              &appH221NonStd);
   }

   else
   {
      gc_util_insert_parm_ref(&pParmBlock,
                              IPSET_NONSTANDARDCONTROL,
                              IPPARM_NONSTANDARDDATA_OBJID,
                              (unsingned char)(strlen(pOid)+1),
                              pOid);
   }
```

Update to the **Retrieving Current Call-Related Information** section

The following note is added under **Section 4.4, Retrieving Current Call-Related Information** (page 73):

Note: When an application on H.323 is using **gc_Extension( )** to extract information from a GCEV_OFFERED event, the application must ensure that it acknowledges the call within 8 seconds to prevent the offering side from timing out. The timer can be extended by sending PROCEEDING (by calling **gc_CallAck( )**) or ALERTING (by calling **gc_AcceptCall( )**) before extracting the information.

Update to the **Retrieving Nonstandard Data From Protocol Messages When Using H.323** section

Section 4.4.1, Retrieving Nonstandard Data From Protocol Messages When Using H.323 (page 76) is replaced by the following information to reflect support for Nonstandard Data longer than 255 bytes:

4.4.1 Retrieving Nonstandard Data From Protocol Messages When Using H.323

Any received Q.931 message can include Nonstandard Data. The application can use the **gc_Extension( )** function with an **ext_id** (extension ID) of IPEXTID_GETINFO to retrieve the data while a call is in any state. The **target_type** should be GCTGT_GCLIB_CRN and the

target_id should be the actual CRN. The information is included with the corresponding GCEV_EXTENSIONCMPLT termination event.

Note: When retrieving nonstandard data, it is only necessary to specify the IPPARM_NONSTANDARDDATA_DATA parameter ID in the extension request. It is not necessary to specify the ID for the nonstandard identifier parameter (that is, IPPARM_NONSTANDARDDATA_OBJID or IPPARM_H221NONSTANDARD). The call control library ensures that the GCEV_EXTENSIONCMPLT event includes the correct information.

When retrieving nonstandard data from the GC_PARM_BLK associated with the metadata of the GCEV_EXTENSIONCMPLT event, it is important to use the extended **gc_util_..._ex( )** functions because the IPPARM_NONSTANDARDDATA_DATA parameter is defined to support data that may be longer than 255 bytes. The actual maximum data length is configured by the application via the max_parm_data_size field in the IPCCLIB_START_DATA structure when it initializes the library; the default size is 255, but the application can set any value up to 4096.

Update to the **Specifying DTMF Support** section

The following note is added to the end of **Section 4.7.1, Specifying DTMF Support** (page 93):

Note: When switching an Intel NetStructure IPT Series board to RFC2833 mode, the change will not take effect unless the payload type is set as well as the DTMF transfer type.

Update to the **Sending Protocol Messages** section

Section 4.10, Sending Protocol Messages (page 97) is replaced by the following information to reflect support for Nonstandard Data longer than 255 bytes:

4.10 Sending Protocol Messages

The Global Call library allows applications that are using the H.323 protocol to send certain messages that contain Nonstandard Data. This capability is supported for the following message types:

- User Input Indication (UII) Message (H.245)
- Facility Messages (Q.931)
- Registration Messages

The following table summarizes the set IDs and parameter IDs used to send the messages and describes the call states in which each message should be sent.

Summary of Protocol Messages that Can be Sent with Nonstandard Data

Type	Set ID & Parameter ID	When Message Should be Sent
Nonstandard UII Message (H.245)	IPSET_MSG_H245 • IPPARM_MSGTYPE value = IP_MSGTYPE_H245_INDICATION	Only when call is in Connected state
Nonstandard Facility Message (Q.931)	IPSET_MSG_Q931 • IPPARM_MSGTYPE value = IP_MSGTYPE_Q931_FACILITY	In any call state
Nonstandard Registration Message	IPSET_MSG_RAS • IPPARM_MSGTYPE value = IP_MSGTYPE_REG_NONSTD	

The maximum length of the Global Call parameter used for the Nonstandard Data information is configured at start-up via the max_parm_data_size field in the IPCCLIB_START_DATA structure. The default size is 255 (for backwards compatibility), but applications may configure it to be as large as 4096 bytes. Applications must use the extended **gc_util_..._ex( )** functions to insert or extract any GC_PARM_BLK parameter elements whose data length is defined to be greater than 255.

Note: In practice, applications may not be able to utilize the full maximum length of the nonstandard data parameter element as configured in max_parm_data_size. The H.323 stack limits the overall size of messages to be max_parm_data_size + 512 bytes, and any messages that exceed this limit are truncated without any notification to the application.

Update to the **Nonstandard UII Message (H.245)** section

Section 4.10.1, Nonstandard UII Message (H.245) (pages 98-99) is replaced by the following information to correct inaccuracies and reflect support for Nonstandard Data longer than 255 bytes:

4.10.1 Nonstandard UII Message (H.245)

To send nonstandard UII messages, use the **gc_Extension( )** function in asynchronous mode with an **ext_id** (extension ID) of IPEXTID_SENDMSG. The **target_type** should be GCTGT_GCLIB_CRN and the **target_id** should be the actual CRN. The GC_PARM_BLK must contain parameter elements that identify the message type, the nonstandard data, and the nonstandard data identifier. At the sending end, reception of a GCEV_EXTENSIONCMPLT event indicates that the message has been sent.

The parameter element that identifies the message type is:

IPSET_MSG_H245
IPPARM_MSGTYPE
- value = IP_MSGTYPE_H245_INDICATION

The parameter element for the Nonstandard Data data is:

IPSET_NONSTANDARDDATA
    IPPARM_NONSTANDARDDATA_DATA
    - value = Nonstandard Data string, max length = max_parm_data_size (configurable at library start-up)

The parameter element for the Nonstandard Data identifier is one (and only one) of the following:

IPSET_NONSTANDARDDATA
    IPPARM_NONSTANDARDDATA_OBJID
    - value = array of unsigned integers, max length = MAX_NS_PARM_OBJID_LENGTH

IPSET_NONSTANDARDDATA
    IPPARM_H221NONSTANDARD
    - value = IP_H221NONSTANDARD structure

When the Global Call library receives a nonstandard UII message, it generates a GCEV_EXTENSION event with the ext_id value IPEXTID_RECEIVEMSG. The extevtdatap field in the METAEVENT structure for the GCEV_EXTENSION event is a pointer to an EXTENSIONEVTBLK structure which in turn contains a GC_PARM_BLK that includes all of the data in the message.

```
.
.
.
/* H245 UII with ObjId and data */

rc = gc_util_insert_parm_val(&t_PrmBlkp, IPSET_MSG_H245, IPPARM_MSGTYPE,
                             sizeof(int), IP_MSGTYPE_H245_INDICATION);

rc = gc_util_insert_parm_ref_ex(&t_PrmBlkp, IPSET_NONSTANDARDDATA,
                                IPPARM_NONSTANDARDDATA_OBJID, ObjLen+1, ObjId);

rc = gc_util_insert_parm_ref_ex(&t_PrmBlkp, IPSET_NONSTANDARDDATA,
                                IPPARM_NONSTANDARDDATA_DATA, DataLen+1, data);

if (rc == -1)
{
   printf("Fail to insert parm");
   return -1;
}
else
   printf("Sending IP H245 UII Message");

gc_Extension(GCTGT_GCLIB_CRN,
             crn,
             IPEXTID_SENDMSG,
             t_PrmBlkp,
             &t_RetBlkp,
             EV_ASYNC);

gc_util_delete_parm(t_PrmBlkp);
.
.
.
```

Update to the **Nonstandard Facility Message (Q.931)** section

Section 4.10.2, Nonstandard Facility Message (Q.931) (pages 99-100) is replaced by the following information to correct inaccuracies and reflect support for Nonstandard Data longer than 255 bytes:

4.10.2 Nonstandard Facility Message (Q.931)

To send a nonstandard Facility message, use the **gc_Extension( )** function in asynchronous mode with an **ext_id** (extension ID) of IPEXTID_SENDMSG. The **target_type** should be GCTGT_GCLIB_CRN and the **target_id** should be the actual CRN. The GC_PARM_BLK must contain parameter elements that identify the message type, the nonstandard data, and the nonstandard data identifier. At the sending end, reception of a GCEV_EXTENSIONCMPLT event indicates that the message has been sent.

The parameter element that identifies the message type is:

IPSET_MSG_Q931
IPPARM_MSGTYPE
- value = IP_MSGTYPE_Q931_FACILITY

The parameter element for the Nonstandard Data data is:

IPSET_NONSTANDARDDATA
IPPARM_NONSTANDARDDATA_DATA
- value = Nonstandard Data string, max length = max_parm_data_size (configurable at library start-up)

The parameter element for the Nonstandard Data identifier is one (and only one) of the following:

IPSET_NONSTANDARDDATA
IPPARM_NONSTANDARDDATA_OBJID
- value = array of unsigned integers, max length = MAX_NS_PARM_OBJID_LENGTH

IPSET_NONSTANDARDDATA
IPPARM_H221NONSTANDARD
- value = IP_H221NONSTANDARD structure

When the Global Call library receives a nonstandard Facility message, it generates a GCEV_EXTENSION event with the ext_id value IPEXTID_RECEIVEMSG. The extevtdatap field in the METAEVENT structure for the GCEV_EXTENSION event is a pointer to an EXTENSIONEVTBLK structure which in turn contains a GC_PARM_BLK that includes all of the data in the message.

The following code shows how to set up and send a Q.931 nonstandard facility message.

```
char ObjId[]= "1 22 333 4444";
char NSData[]= "DataField_Facility";

GC_PARM_BLKP    gcParmBlk = NULL;
```

```
gc_util_insert_parm_val(&gcParmBlk,
                        IPSET_MSG_Q931,
                        IPPARM_MSGTYPE,
                        sizeof(int),
                        IP_MSGTYPE_Q931_FACILITY);

gc_util_insert_parm_ref(&gcParmBlk,
                        IPSET_NONSTANDARDDATA,
                        IPPARM_NONSTANDARDDATA_OBJID,
                        sizeof(ObjId),
                        ObjId);

gc_util_insert_parm_ref_ex(&gcParmBlk,
                           IPSET_NONSTANDARDDATA,
                           IPPARM_NONSTANDARDDATA_DATA,
                           sizeof(NSData),
                           NSData);

gc_Extension( GCTGT_GCLIB_CRN,
              crn,
              IPEXTID_SENDMSG,
              gcParmBlk,
              NULL,
              EV_ASYNC);

gc_util_delete_parm_blk(gcParmBlk);
```

Update to the **Nonstandard Registration Message** section

Section 4.10.3, Nonstandard Registration Message (pages 100-101) is replaced by the following information to correct inaccuracies and reflect support for Nonstandard Data longer than 255 bytes:

4.10.3 Nonstandard Registration Message

To send a nonstandard registration message, use the **gc_Extension( )** function in asynchronous mode with an **ext_id** (extension ID) of IPEXTID_SENDMSG. The **target_type** should be GCTGT_CCLIB_NETIF and the **target_id** should be the board device handle, since the message destination is the Gatekeeper. The GC_PARM_BLK must contain parameter elements that identify H.323 protocol, the message type, the nonstandard data, and the nonstandard data identifier. The application receives a GCEV_EXTENSIONCMPLT event to indicate that the message has been sent.

The following parameter element sets the protocol to be H.323:

IPSET_PROTOCOL
- IPPARM_PROTOCOL_BITMASK
  - value = IP_PROTOCOL_H323

The parameter element that identifies the message type is:

IPSET_MSG_REGISTRATION
- IPPARM_MSGTYPE
  - value = IP_MSGTYPE_REG_NONSTD

The parameter element for the Nonstandard Data data is:

IPSET_NONSTANDARDDATA
IPPARM_NONSTANDARDDATA_DATA
- value = Nonstandard Data string, max length = max_parm_data_size (configurable at library start-up)

The parameter element for the Nonstandard Data identifier is one (and only one) of the following:

IPSET_NONSTANDARDDATA
IPPARM_NONSTANDARDDATA_OBJID
- value = array of unsigned integers, max length = MAX_NS_PARM_OBJID_LENGTH

IPSET_NONSTANDARDDATA
IPPARM_H221NONSTANDARD
- value = IP_H221NONSTANDARD structure

The following code snippet illustrates how to send an H.323 nonstandard registration message.

```
{
   GC_PARM_BLKP parmblkp = NULL;
   char h221nonstd_id[] = "My H.221 Nonstandard data identifier";
                                       /* must be <= MAX_NS_PARM_OBJID_LENGTH (40) */
   char nonstd_data[] = "My nonstandard_data";

   gc_util_insert_parm_val(&parmblkp, IPSET_PROTOCOL, IPPARM_PROTOCOL_BITMASK,
                           sizeof(char), IP_PROTOCOL_H323);
   gc_util_insert_parm_val(&parmblkp, IPSET_MSG_REGISTRATION, IPPARM_MSGTYPE,
                           sizeof(unsigned long), IP_MSGTYPE_REG_NONSTD);
   gc_util_insert_parm_ref_ex(&parmblkp, IPSET_NONSTANDARDDATA, IPPARM_NONSTANDARDDATA_DATA,
                              sizeof(nonstd_data), nonstd_data);
   gc_util_insert_parm_ref(&parmblkp, IPSET_NONSTANDARDDATA, IPPARM_H221NONSTANDARD,
                           sizeof(h221nonstd_id), h221nonstd_id);

   if (gc_Extension(GCTGT_CCLIB_NETIF, bdev, IPEXTID_SENDMSG, parmblkp, NULL,
                    EV_ASYNC) != GC_SUCCESS)
   {
      printandlog(ALL_DEVICES, GC_APIERR, NULL, "gc_Extension() Failed", 0);
      exitdemo(1);
   }
}
```

Update to the **Quality of Service Alarm Management** section

In **Section 4.15, Quality of Service Alarm Management**, the first bullet item (page 104) is changed to read:

- lost packets (Intel NetStructure IPT Series only)

Update to the **Setting QoS Threshold Values** section

The code example in **Section 4.15.3, Setting QoS Threshold Values** (page 105), is updated to use the Jitter alarm, since that is the only QoS alarm type supported on both DM/IP and IPT boards. The initialization lines now read:

```
ALARM_PARM_LIST alarm_parm_list;
IPM_QOS_THRESHOLD_INFO QoS_info;
alarm_parm_list.n_parms = 1;
QoS_info.unCount=1;
QoS_info.QoSThresholdData[0].eQoSType = QOSTYPE_JITTER;
QoS_info.QoSThresholdData[0].unTimeInterval = 1000;
QoS_info.QoSThresholdData[0].unDebounceOn = 5000;
```

```
QoS_info.QoSThresholdData[0].unDebounceOff = 15000;
QoS_info.QoSThresholdData[0].unFaultThreshold = 50;
QoS_info.QoSThresholdData[0].unPercentSuccessThreshold = 90;
QoS_info.QoSThresholdData[0].unPercentFailThreshold = 10;
```

Update to the **Retrieving QoS Threshold Values** section

A new note is added before the code example in **Section 4.15.4, Retrieving QoS Threshold Values** (page 106):

3. The Lost Packets QoS alarm is only supported when using Intel NetStructure IPT Series boards.

Update to the **gc_MakeCall( ) Variances for IP** section (Defect# IPY00029956 = PTR# 36646)

The last paragraph before **Section 7.2.16.1** (page 172) is updated as follows:

When using SIP, if the remote side does not send a final response to an outgoing INVITE (sent by the call control library) within 32 seconds, the **gc_MakeCall( )** function times out and the library generates a GCEV_DISCONNECTED event to the application. If the application attempts to drop the call before the 32 second time-out is reached, the library's behavior depends on whether a provisional response was received. When no provisional response was received before the application cancels the call, the library cleans up the call immediately. But if a provisional response was received before the application attempts to cancel the call, the library sends a CANCEL to the remote endpoint and generates a GCEV_DROPCALL to the application after it receives the 200OK response to the CANCEL and a 487RequestTerminated response for the original INVITE, or when an additional 32-second time-out expires.

Update to the **gc_OpenEx( ) Variances for IP** section

The following paragraph should be added to **Section 7.2.17, gc_OpenEx( ) Variances for IP** (pages 188-189). This paragraph should also be considered to be a variance for **gc_Close( )**, which does not have a “Variances for IP” section in this edition of the document. (PTR# 32087)

- Applications should avoid closing and re-opening devices multiple times. Board devices and channel devices should be opened during initialization and should remain open for the duration of the application.

Update to the **gc_SetAlarmParms( ) Variances for IP** section

In **Section 7.2.23, gc_SetAlarmParms( ) Variances for IP**, the list of supported alarm threshold types in the description of **alarm_parm_list** (page 196) should note that QOSTYPE_LOSTPACKET is only supported for IPT Series boards.

Update to the **IPSET_NONSTANDARDCONTROL** section

Section 8.2.15, IPSET_NONSTANDARDCONTROL (page 224) is replaced by the following information to reflect support for Nonstandard Control data longer than 255 bytes:

8.2.15 IPSET_NONSTANDARDCONTROL

The following table shows the parameter IDs in the IPSET_NONSTANDARDCONTROL parameter set.

IPSET_NONSTANDARDCONTROL Parameter Set

Parameter IDs	Data Type & Size	Description	SIP/ H.323
IPPARM_ NONSTANDARDDATA_DATA	Type: string † Size: max. length = max_parm_data_size ‡ (configured at start-up via IPCCLIB_START_DATA)	Used to contain the nonstandard data.	H.323 only
IPPARM_ NONSTANDARDDATA_OBJID	Type: Uint[ ] Size: max. length = MAX_NS_PARM_ OBJID_LENGTH (40)	Used to contain a nonstandard object ID, if any. If an H.221 nonstandard data identifier is being used, this parameter should not be present in the parm block.	H.323 only
IPPARM_H221NONSTANDARD	Type: IP_H221NONSTANDARD Size: sizeof(IP_H221NONSTANDARD)	Used to contain a H.221 nonstandard data identifier, if any. If a nonstandard object ID is being used, this parameter should not be present in the parm block.	H.323 only

† For parameters with data of type String, the length of in a GC_PARM_BLK is the length of the data string plus 1.
‡ The full maximum length that is configured may not be usable in practice because the H.323 stack limits total message size to max_parm_data_size + 512 bytes. Longer messages are truncated without notification to the application.

Update to the **IPSET_NONSTANDARDDATA** section

Section 8.2.16, IPSET_NONSTANDARDDATA (page 224) is replaced by the following information to reflect support for Nonstandard Data longer than 255 bytes:

8.2.16 IPSET_NONSTANDARDDATA

The following table shows the parameter IDs in the IPSET_NONSTANDARDDATA parameter set.

IPSET_NONSTANDARDDATA Parameter Set

Parameter IDs	Data Type & Size	Description	SIP/ H.323
IPPARM_ NONSTANDARDDATA_DATA	Type: string † Size: max. length = max_parm_data_size ‡ (configured at start-up via IPCCLIB_START_DATA)	Used to contain the nonstandard data.	H.323 only
IPPARM_ NONSTANDARDDATA_OBJID	Type: Uint[ ] Size: max. length = MAX_NS_PARM_OBJID_ LENGTH (40)	Used to contain a nonstandard object ID, if any. If an H.221 nonstandard data identifier is being used, this parameter should not be present in the parm block.	H.323 only
IPPARM_H221NONSTANDARD	Type: IP_H221NONSTANDARD Size: sizeof(IP_H221NONSTANDARD)	Used to contain an H.221 nonstandard data identifier, if any. If a nonstandard object ID is being used, this parameter should not be present in the parm block.	H.323 only
† For parameters with data of type String, the length in a GC_PARM_BLK is the length of the data string plus 1. ‡ The full maximum length that is configured may not be usable in practice because the H.323 stack limits total message size to max_parm_data_size + 512 bytes. Longer messages are truncated without notification to the application.			

Update to the **IP_H221NONSTANDARD** data structure

On the reference page for the IP_H221NONSTANDARD data structure (page 238), the descriptions of the three data fields are updated as follows:

country_code

The country code. Range: 0 to 255; any value x>255 is treated as x%256.

extension

The extension number. Range: 0 to 255; any value x>255 is treated as x%256.

manufacturer_code

The manufacturer code. Range: 0 to 65535; any value x>65535 is treated as x%65636.

Update to the **IP_VIRTBOARD** data structure

The typedef and descriptive text for the IP_VIRTBOARD data structure (pages 240-241) are missing three new fields and indicate the wrong name for the data type of the localIP field. There have also been changes to the IP_VIRTBOARD data structure because of new features introduced in the Service Update. The information on the IP_VIRTBOARD structure is updated as shown below.

IP_VIRTBOARD

```
typedef struct
{
    unsigned short    version;
    unsigned int      total_max_calls;
    unsigned int      h323_max_calls;
    unsigned int      sip_max_calls;
    IP_ADDR           localIP;
    unsigned short    h323_signaling_port;
    unsigned short    sip_signaling_port;
    void              *reserved;
    unsigned short    size;
    unsigned int      sip_msginfo_mask;
    unsigned int      sup_serv_mask;
    unsigned int      h323_msginfo_mask;
    MIME_MEM          sip_mime_mem
    unsigned short    terminal_type
    IP_ADDR           outbound_proxy_IP
    unsigned short    outbound_proxy_port;
    char *            outbound_proxy_hostname;

}IP_VIRTBOARD;
```

Description

The IP_VIRTBOARD data structure is used to store configuration and capability information about an IPT board device that is used when the device is started. An array of IP_VIRTBOARD structures (one for each virtual board in the system) is referenced by the IPCCLIB_START_DATA structure, which is passed to the **gc_Start( )** function. The IP_VIRTBOARD structure must be initialized to default values by the **INIT_IP_VIRTBOARD( )** initialization function; those default values can be overridden by the application before calling **gc_Start( )**.

Field Descriptions

The fields of the IP_VIRTBOARD data structure are described as follows:

version
: The version of the structure. The correct version number is populated by the **INIT_IP_VIRTBOARD( )** function and should not be overridden by the application.

total_max_calls
: The maximum total number of IPT devices that can be open concurrently using either the H.323 or SIP protocol. Valid values range from 1 to IP_CFG_MAX_AVAILABLE_CALLS (=2016). The default value is 120. This field must not be set to IP_CFG_NO_CALLS (=0) and must not be set to a value larger than the sum of h323_max_calls and sip_max_calls.

h323_max_calls
: The maximum number of IPT devices that can be used for H.323 calls. Valid values are in the range IP_CFG_NO_CALLS (=0) to IP_CFG_MAX_AVAILABLE_CALLS (=2016). The default value is 120. This field must not be set to IP_CFG_NO_CALLS if sip_max_calls is also set to that value.

sip_max_calls
: The maximum number of IPT devices that can be used for SIP calls. Possible values are in the range IP_CFG_NO_CALLS (=0) to IP_CFG_MAX_AVAILABLE_CALLS (=2016). The

default value is 120. This field must not be set to IP_CFG_NO_CALLS if h323_max_calls is also set to that value.

localIP
The local IP address of type IP_ADDR. See the reference page for the IP_ADDR data structure for more information.

h323_signaling_port
The H.323 call signaling port. Possible values are a valid port number or IP_CFG_DEFAULT. The default H.323 signaling port is 1720.

sip_signaling_port
The SIP call signaling port. Possible values are a valid port number or IP_CFG_DEFAULT. The default SIP signaling port is 5060.

reserved
For library use only

size
For library use only

sip_msginfo_mask (structure version ≥ 0x101 only)
Enables and disables access to SIP message information. Access is disabled by default. The following mask values, which may be OR'ed together, are defined to enable these features:

- IP_SIP_MSGINFO_ENABLE – enable access to supported SIP message information fields
- IP_SIP_MIME_ENABLE – enable sending and receiving of SIP messages that contain MIME information
- IP_SIP_FASTSTART_CODERS_IN_OFFERED – enable receiving coder information from a SIP "FastStart" call offer via the GCEV_OFFERED event

sup_serv_mask (structure version ≥ 0x102 only)
Enables and disables the call transfer supplementary service. The service is disabled by default. Use the following value to enable the feature:

- IP_SUP_SERV_CALL_XFER – enable call transfer service

h323_msginfo_mask (structure version ≥ 0x103 only)
Enables and disables reception of H.323 message information. Access is disabled by default. The following mask values, which may be OR'ed together, are defined to enable the features:

- IP_H323_ANNEXMMSG_ENABLE – enable reception of H.323 Annex M tunneled signaling messages in H.225 messages
- IP_H323_MSGINFO_ENABLE – enable access to H.323 message information fields
- IP_H323_FASTSTART_CODERS_IN_OFFERED – enable receiving coder information from an H.323 "FastStart" call offer via the GCEV_OFFERED event

sip_mime_mem (structure version ≥ 0x104 only)
Sets the number and size of buffers that will be allocated for the MIME memory pool when the SIP MIME feature is enabled (no buffers are allocated if the feature is not enabled). The default values indicated below are set by the **INIT_MIME_MEM( )** macro, which is called by the **INIT_IP_VIRTBOARD( )** initialization function. The MIME_MEM data structure is defined as follows:

```
typedef struct
{
     unsigned short   version;     /* Version set by INIT_MIME_MEM   */
     unsigned int     size;        /* Default = 1500                 */
     unsigned int     number;      /* Default = (sip_max_calls * 5)  */
}MIME_MEM;
```

terminal_type (structure version ≥ 0x104 only)

Sets the Terminal Type for the virtual board which will be used during RAS registration (H.323 terminal type) and during Master Slave determination (H.245 terminal type). The value may only be changed from the default that is set by the **INIT_IP_VIRTBOARD( )** initialization function before calling **gc_Start( )**. Unsigned shorts from 0 to 255 are valid values, but the specific values 0 and 255 are reserved and will result in the terminal type being set to the default. Values larger than 255 are truncated to 8 bits. The following symbolic values are defined:

- IP_TT_GATEWAY (default) – value = 60, for operation as terminal type Gateway
- IP_TT_TERMINAL – value = 50, for operation as terminal type Terminal

outbound_proxy_IP (structure version ≥ 0x105 only)

Sets the IP address of the SIP outbound proxy, which is used instead of the original Request URI for outbound SIP requests. The default value is 0, which disables outbound proxy unless the outbound_proxy_hostname field is set to a non-NULL name.

outbound_proxy_port (structure version ≥ 0x105 only)

Sets the port number of the SIP outbound proxy specified by outbound_proxy_IP. The default value is 5060, which is the same as the default SIP signaling port number.

outbound_proxy_hostname (structure version ≥ 0x105 only)

Sets the specified hostname as the SIP outbound proxy instead of a hard-coded IP address. If outbound_proxy_IP is set to 0, this hostname is resolved as the outbound proxy address. If outbound_proxy_IP is set to an IP address, this field is ignored and outbound_proxy_IP and outbound_proxy_port are used instead. The default value is NULL.

Update to the **IP_ADDR** data structure

The data structure reference information for **IP_ADDR** (page 242) identifies the structure by an incorrect name (namely **IPADDR**). The typedef for the structure is updated as follows, but the field descriptions are accurate as written:

```
typedef struct
{
    unsigned char     ip_ver;
    union
    {
        unsigned int      ipv4;
        unsigned int      ipv6[4]
    }u_ipaddr;
}IP_ADDR, *IP_ADDRP;
```

Update to the **IPCCLIB_START_DATA** data structure

The data structure reference information for IPCCLIB_START_DATA (page 243) is updated to reflect the addition of a field to support data in specific Global Call parameter elements that is longer than 255 bytes. The information on the IPCCLIB_START_DATA structure is updated as shown below.

IPCCLIB_START_DATA

```
typedef struct
{
    unsigned short    version;
    unsigned char     delimiter;
    unsigned char     num_boards;
    IP_VIRTBOARD      *board_list;
    unsigned long     max_parm_data_size;
} IPCCLIB_START_DATA;
```

Description

The IPCCLIB_START_DATA structure is used to configure the IP call control library when starting Global Call. The IPCCLIB_START_DATA structure is passed to the **gc_Start( )** function via the CCLIB_START_STRUCT and GC_START_STRUCT data structures. Applications must use the **INIT_IPCCLIB_START_DATA( )** function to populate a IPCCLIB_START_DATA structure with default values before overriding the default values as desired.

Field Descriptions

The fields of the IPCCLIB_START_DATA data structure are described as follows:

version
: The version of the start structure. The correct version number is populated by the **INIT_IPCCLIB_START_DATA( )** function and should not be used by applications.

delimiter
: An ANSI character that specifies the address string delimiter; the default delimiter is the comma (,). The specified delimiter character is used to separate the components of the destination information when using **gc_MakeCall( )**, for example.

num_boards
: The number of IPT virtual board devices to create. Maximum value is 8; default value is 2.

board_list
: A pointer to an array of IP_VIRTBOARD structures, one structure for each of num_boards IPT board devices.

max_parm_data_size (structure version ≥ 0x200)
: The maximum data size (in bytes) for Global Call parameters that support data lengths greater than 255 bytes. The default value for this field is 255 for backwards compatibility; the maximum value is 4096.

  Only specific Global Call parameters support >255 byte data. These parameters include:

  - IPSET_NONSTANDARDCONTROL / IPPARM_NONSTANDARDDATA_DATA
  - IPSET_NONSTANDARDDATA / IPPARM_NONSTANDARDDATA_DATA
  - IPSET_TUNNELEDSIGNALMSG / IPPARM_TUNNELEDSIGNALMSG_DATA

  Note: When using H.323, the stack limits the total size of messages to 4096 bytes, so that it may not be possible to use the maximum parameter data size defined in this field for Nonstandard Data or Tunneled Signaling Message data due to the presence of other payload in the message. If the total size of an H.323 message is greater than 4096 bytes, the stack truncates the message with no notification to the application.

New utility functions

The following four utility functions have been added to the Global Call library to support data that is longer than 255 bytes in Global Call parameter elements.

- **gc_util_copy_parm_blk( )**
- **gc_util_find_parm_ex( )**
- **gc_util_insert_parm_ref_ex( )**
- **gc_util_next_parm_ex( )**

These functions are backwards compatible and may be used with standard parameter elements as well as parameter elements that support extended-length data, but they **must** be used whenever a GC_PARM_BLK might contain parameter elements with extended-length data.

Function reference information for the four new functions is provided below.

gc_util_copy_parm_blk( )

Name: int gc_util_copy_parm_blk(parm_blkpp, parm_blkp)

Inputs: GC_PARM_BLKP* parm_blkpp • pointer to the address of the new GC_PARM_BLK

GC_PARM_BLKP parm_blkp • pointer to a valid GC_PARM_BLK to be copied

Returns: GC_SUCCESS if successful
GC_FAIL if unsuccessful

Includes: gclib.h
gcerr.h

Category: GC_PARM_BLK utility

Mode: synchronous

Description

The **gc_util_copy_parm_blk( )** function copies the specified GC_PARM_BLK.

This function **must** be used to copy any GC_PARM_BLK that contains any setID/parmID pairs that can have data that is potentially larger than 255 bytes. This function can be used for any GC_PARM_BLK, regardless of whether it contains setID/parmID pairs that support parameter data lengths greater than 255 bytes.

Only specific Global Call parameters support >255 byte data and therefore require the use of this function. These parameters include:

- IPSET_NONSTANDARDCONTROL / IPPARM_NONSTANDARDDATA_DATA
- IPSET_NONSTANDARDDATA / IPPARM_NONSTANDARDDATA_DATA
- IPSET_TUNNELEDSIGNALMSG / IPPARM_TUNNELEDSIGNALMSG_DATA

Parameter	Description
parm_blkpp	pointer to the address of the new GC_PARM_BLK that the specified parm block will be copied to; **must** be set to NULL
parm_blkp	points to a valid, existing GC_PARM_BLK to be copied

Cautions

To avoid a memory leak, any GC_PARM_BLK created must eventually be deleted using the **gc_util_delete_parm_blk( )** function.

Errors

If this function returns GC_ERROR(-1) to indicate failure, use the **gc_ErrorInfo( )** function to retrieve the reason for the error. See the "Error Handling" section in the *Global Call API Programming Guide*. All Global Call error codes are defined in the *gcerr.h* file.

Example

```
#include "gclib.h"
#include "gcip.h"

void process_event(void)
{
   METAEVENT  metaevent;
   GC_PARM_BLKP my_blkp = NULL;

   if(gc_GetMetaEvent(&metaevent) != GC_SUCCESS)
   {
      /* process error */
   }

   Switch(metaevent.evttype)
   {
      case GCEV_OFFERED:
         /* make a copy of the parm blk */
         if(metaevent.extevtdatap)
         {
            if ( gc_util_copy_parm_blk( &my_blkp,(GC_PARM_BLKP)(metaevent.extevtdatap))
                 != GC_SUCCESS )
            {
               /* Process error */
            }
         }
      ...
   }
   ...
}
```

See Also

- **gc_util_delete_parm_blk( )** (in *Global Call API Library Reference*)

gc_util_find_parm_ex( )

Name: int gc_util_find_parm_ex(parm_blk, setID, parmID, parm)

Inputs:	GC_PARM_BLKP parm_blk	• pointer to GC_PARM_BLK to search for the parameter
	unsigned long setID	• parameter set ID of parameter to be found
	unsigned long parmID	• parameter ID of parameter to be found
	GC_PARM_DATA_EXTP parm	• pointer to a valid GC_PARM_DATA_EXT structure that identifies where in the parm block to start searching
Outputs:	GC_PARM_DATA_EXTP parm	• if successful, pointer to a GC_PARM_DATA_EXT structure that contains the ID and value data for the specified parameter

Returns: GC_SUCCESS if successful
EGC_NO_MORE_PARMS if no more parameters exist in GC_PARM_BLK
GC_ERROR if failure

Includes: gclib.h
gcerr.h

Category: GC_PARM_BLK utility

Mode: synchronous

Description

The **gc_util_find_parm_ex( )** function is used to find a parameter of a particular type in a GC_PARM_BLK and retrieve the parameter data into a GC_PARM_DATA_EXT structure.

This function **must** be used instead of the similar **gc_util_find_parm( )** function if the parameter data can potentially exceed 255 bytes. This function is backward compatible and can be used instead of **gc_util_find_parm( )** for any GC_PARM_BLK, regardless of whether the parameter block contains setID/parmID pairs that support data lengths greater than 255 bytes.

Only specific Global Call parameters support >255 byte data and therefore require the use of this function. These parameters include:

- IPSET_NONSTANDARDCONTROL / IPPARM_NONSTANDARDDATA_DATA
- IPSET_NONSTANDARDDATA / IPPARM_NONSTANDARDDATA_DATA
- IPSET_TUNNELEDSIGNALMSG / IPPARM_TUNNELEDSIGNALMSG_DATA

The **gc_util_find_parm_ex( )** function can be used to determine whether a particular parameter exists, or to retrieve a particular parameter, or both. If the specified parameter is found in the GC_PARM_BLK, the function fills in the GC_PARM_DATA_EXT structure with the parameter data and returns GC_SUCCESS. If the parameter does not exist in the GC_PARM_BLK, or if no more parameters of the specified type are found, the function returns EGC_NO_MORE_PARMS.

To search from the beginning of the GC_PARM_BLK, initialize the GC_PARM_DATA_EXT structure (**parm**) by using **INIT_GC_PARM_DATA_EXT(parm)** before calling **gc_util_find_parm_ex( )**. If the structure pointed to by **parm** contains parameter information that

was retrieved in a previous call to this function, the function will begin its search at that parameter rather than the beginning of the parameter block.

Parameter	Description
parm_blk	points to a valid GC_PARM_BLK that will be searched for a parameter of the specified type
setID	set ID of the parameter to be found
parmID	parameter ID of the parameter to be found
parm	points to a valid GC_PARM_DATA_EXT provided by the application. If a pointer to a newly initialized structure is passed in the function call, the function searches from the beginning of the GC_PARM_BLK; if the structure contains data from a previously found parameter, the function searches from that parameter onward. When the function completes successfully, the structure is updated to contain retrieved information for the parameter that was found.

Cautions

Unlike the similar **gc_util_find_parm( )** function, the **parm** pointer used in this function *cannot* be used to update the parameter itself because it points to a data structure that is in the application's memory rather than a location in the GC_PARM_BLK itself.

Errors

If this function returns GC_ERROR to indicate failure, use the **gc_ErrorInfo( )** function to retrieve the reason for the error. See the "Error Handling" section in the *Global Call API Programming Guide*. All Global Call error codes are defined in the *gcerr.h* file.

Example

```
#include "gclib.h"
#include "gcip.h"

void search_parm_block(GC_PARM_BLKP parm_blkp)
{
   GC_PARM_DATA_EXT parm_data_ext;
   int ret = 0;

   /* Initialize this structure for two reasons:
    * 1. To search from the first parameter in the parm block
    * 2. The first time this structure is used it must be initialized
    */
   INIT_GC_PARM_DATA_EXT(&parm_data_ext);

   /* loop to retrieve all of the parameters and associated data in the
    * GC_PARM_BLK that match the set_ID/parm_ID pair for SIP header fields.
    */
   while ( GC_SUCCESS == (ret = gc_util_find_parm_ex(parm_blkp, IPSET_SIP_MSGINFO,
                                                     IPPARM_SIP_HDR, &parm_data_ext)) )
   {
      /* process GC_PARM_DATA_EXT structure */
      .
      .
      .
   }
```

```
    /* Check for error */
    if ( GC_ERROR == ret)
    {
        /* process error */
    }

    .
    .
    .
}
```

See Also

- **gc_util_find_parm( )** (in *Global Call API Library Reference*)
- **gc_util_next_parm_ex( )**

gc_util_insert_parm_ref_ex( )

Name: int gc_util_insert_parm_ref_ex(parm_blkpp, setID, parmID, data_size, datap)

Inputs:	GC_PARM_BLKP *parm_blkpp	• pointer to the address of a valid GC_PARM_BLK
	unsigned long setID	• set ID of parameter to be inserted
	unsigned long parmID	• parm ID of parameter to be inserted
	unsigned long data_size	• size in bytes of the parameter data
	void *datap	• pointer to the parameter data

Returns: GC_SUCCESS if successful
GC_ERROR if failure

Includes: gclib.h
gcerr.h

Category: GC_PARM_BLK utility

Mode: synchronous

Description

The **gc_util_insert_parm_ref_ex( )** function inserts a parameter into a GC_PARM_BLK data structure by reference.

The **gc_util_insert_parm_ref_ex( )** function **must** be used rather than the similar **gc_util_insert_parm_ref( )** function whenever the parameter data exceeds 255 bytes in length. The **gc_util_insert_parm_ref_ex( )** function is backwards compatible and can be used with any setID/parmID pair regardless of whether that pair supports data lengths greater than 255 bytes.

Only specific Global Call parameters support >255 byte data and therefore require the use of this function. These parameters include:

- IPSET_NONSTANDARDCONTROL / IPPARM_NONSTANDARDDATA_DATA
- IPSET_NONSTANDARDDATA / IPPARM_NONSTANDARDDATA_DATA
- IPSET_TUNNELEDSIGNALMSG / IPPARM_TUNNELEDSIGNALMSG_DATA

A new GC_PARM_BLK can be created by inserting the first parameter with ***parm_blkpp** set to NULL. A parameter can be inserted in an existing GC_PARM_BLK by setting ***parm_blkpp** to the address of that block.

Note: Parameters are contained in the GC_PARM_BLK in the order in which they are inserted, and they will also be retrieved via the **gc_util_next_parm_ex( )** function in the same order.

Parameter	Description
parm_blkpp	points to the address of a valid GC_PARM_BLK where the parameter is to be inserted. Set ***parm_blkpp** to NULL to insert the parameter into a new block.
setID	parameter set ID of the parameter to be inserted

Parameter	Description
parmID	parameter ID of the parameter to be inserted
data_size	size, in bytes, of the data associated with this parameter
datap	points to the data associated with this parameter

Cautions

- To avoid a memory leak, any GC_PARM_BLK created must be deleted using the **gc_util_delete_parm_blk( )** function.
- Insertion of data that exceeds 255 bytes in length is only supported for specific setID/parmID pairs. Refer to the appropriate Global Call Technology Guide for information on maximum data length for each setID/parmID pair.

Errors

- If this function returns GC_ERROR to indicate failure, use the **gc_ErrorInfo( )** function to retrieve the reason for the error. See the "Error Handling" section in the *Global Call API Programming Guide*. All Global Call error codes are defined in the *gcerr.h* file.
- Attempting to insert data greater than 255 bytes in length using a setID/parmID pair that does not support large data sizes produces an error indication. In this situation, the **gc_ErrorInfo( )** function returns the value EGC_INVPARM.

Example

```
#include "gclib.h"
#include "gcip.h"

void SetHeader(void)
{
   GC_PARM_BLKP my_blkp = NULL;
   char* pChar = "Remote-Party_ID: This string can be greater than 255 bytes";

   /* Add 1 to strlen for null termination */
   unsigned long data_size = strlen(pChar) + 1;

   /* insert parm and associated data into the GC_PARM_BLK */
   if ( gc_util_insert_parm_ref_ex( &my_blkp, IPSET_SIP_MSGINFO, IPPARM_SIP_HDR, data_size,
                                    (void*)( pChar )) != GC_SUCCESS )
   {
      /* Process error */
   }

   /* At this point the application can overwrite the data pointed to by pChar. */
   pChar = NULL;

   /* Pass the parm block to GC */
   if ( gc_SetUserInfo( GCTGT_GCLIB_CRN, crn, &my_blkp, GC_SINGLECALL) != GC_SUCCESS )
   {
      /* Process error */
   }

   /* GC_PARM_BLK is no longer needed; delete the block */
   gc_util_delete_parm_blk( my_blkp );
}
```

See Also

- **gc_util_delete_parm_blk( )** (in *Global Call API Library Reference*)
- **gc_util_insert_parm_ref( )** (in *Global Call API Library Reference*)
- **gc_util_insert_parm_val( )** (in *Global Call API Library Reference*)

gc_util_next_parm_ex( )

Name: int gc_util_next_parm_ex(parm_blk, parm)

Inputs: GC_PARM_BLKP parm_blk • pointer to GC_PARM_BLK

GC_PARM_DATA_EXTP parm • pointer to structure identifying current parameter

Outputs: GC_PARM_DATA_EXTP parm • pointer to structure identifying next parameter

Returns: GC_SUCCESS if successful
EGC_NO_MORE_PARMS if no more parameters exist in the GC_PARM_BLK
GC_ERROR if failure

Includes: gclib.h
gcerr.h

Category: GC_PARM_BLK utility

Mode: synchronous

Description

The **gc_util_next_parm_ex( )** function is used to retrieve the next parameter element (relative to a specified current parameter element) from a GC_PARM_BLK in the form of a GC_PARM_DATA_EXT data structure. Calling this function repetitively and passing a pointer to the GC_PARM_DATA_EXT structure that was returned by the previous call allows an application to sequentially retrieve all of the parameter elements in a GC_PARM_BLK. To begin retrieving parameter elements at the beginning of the GC_PARM_BLK, the application initializes the GC_PARM_DATA_EXT structure by calling **INIT_GC_PARM_DATA_EXT(parm)**.

This function **must** be used instead of **gc_util_next_parm( )** if the parameter data can potentially exceed 255 bytes. This function is backward compatible and can be used instead of **gc_util_next_parm( )** for any GC_PARM_BLK, regardless of whether the parameter block contains setID/parmID pairs that support data lengths greater than 255 bytes.

Only specific Global Call parameters support >255 byte data and therefore require the use of this function. These parameters include:

- IPSET_NONSTANDARDCONTROL / IPPARM_NONSTANDARDDATA_DATA
- IPSET_NONSTANDARDDATA / IPPARM_NONSTANDARDDATA_DATA
- IPSET_TUNNELEDSIGNALMSG / IPPARM_TUNNELEDSIGNALMSG_DATA

The **gc_util_next_parm_ex( )** function updates the data structure referenced by the **parm** pointer and returns GC_SUCCESS if there is another parameter element in the GC_PARM_BLK following the current element that was identified in the function call. If the current parameter data structure referenced by **parm** identifies the last parameter element in the GC_PARM_BLK, the next function call returns EGC_NO_MORE_PARMS.

Parameter	Description
parm_blk	points to the valid GC_PARM_BLK structure where data is stored
parm	pointer to a valid GC_PARM_DATA_EXT structure provided by the application. If the pointer that is passed in the function call refers to a structure that was just initialized with **INIT_GC_PARM_DATA_EXT(parm)**, the function retrieves the first parameter element in the GC_PARM_BLK. If the passed pointer references a structure that contains data from a previously found parameter element, the function retrieves the next parameter element in the block (if any). When the function completes successfully, the GC_PARM_DATA_EXT structure is updated to contain the retrieved information for the parameter element.

Cautions

Unlike the similar **gc_util_next_parm( )** function, the **parm** pointer used in this function *cannot* be used to update the parameter itself because it references a data structure that is in the application's memory rather than point to a location within the GC_PARM_BLK itself.

Errors

If this function returns GC_ERROR to indicate failure, use the **gc_ErrorInfo( )** function to retrieve the reason for the error. See the "Error Handling" section in the *Global Call API Programming Guide*. All Global Call error codes are defined in the *gcerr.h* file.

Example

```
#include "gclib.h"
#include "gcip.h"

void process_parm_block(GC_PARM_BLKP pparm_blk)
{
   GC_PARM_DATA_EXT parm_data_ext;
   int ret = 0;

   /* Initialize this structure for two reasons:
    * 1. To retrieve the first parameter in the parm block
    * 2. The first time this structure is used it must be initialized
    */
   INIT_GC_PARM_DATA_EXT(&parm_data_ext);

   /* Loop to retrieve all of the parameters and associated data from the GC_PARM_BLK
    */
   while ( GC_SUCCESS == (ret = gc_util_next_parm_ex( pparm_blk, &parm_dat_ext)) )
   {
      /* Process set_ID/parm_ID pairs */
      switch(parm_data_ext.set_ID);
      {
         .
         .
         .
      }
   }
```

```
    /* Check for error */
    if ( GC_ERROR == ret )
    {
       /* Process error */
    }

    .
    .
    .
}
```

See Also

- **gc_util_find_parm_ex( )**
- **gc_util_next_parm( )** (in *Global Call API Library Reference*)

3.4.11 Global Call ISDN Technology Guide

Update to **Chapter 2, Global Call Architecture for ISDN** (IPY00006540 = PTR# 34211)
The following information should be appended to the end of Chapter 2:

GCEV_EXTENSION Events

There are ISDN-specific Global Call events, which will eventually be mapped to GCEV_EXTENSION. But to maintain backward compatibility, the Global Call application has the option to choose ISDN-specific events or GCEV_EXTENSION. The default is ISDN-specific events. For more information, refer to Section 4.2, “Operations Performed Using RTCM”.

Note: When using DM3 boards, the GCEV_EXTENSION event is not supported. DM3 boards use ISDN-specific events only.

If the application needs to use the new generic call model or extension features, **gc_Start( )** should be called as shown below:

```
CCLIB_START_STRUCT cclib_struct;
GC_START_STRUCT gc_start_struct;
GC_PARM_BLK *parmblk = NULL;

gc_util_insert_parm_val( &parmblk,
                         GCIS_SET_GENERIC,
                         GCIS_PARM_EXTENSIONEVENT,
                         sizeof( char ), 1);

gc_util_insert_parm_val( &parmblk,
                         GCIS_SET_GENERIC,
                         GCIS_PARM_GENERICCALLMODEL,
                         sizeof( char ), 1);
gc_start_struct.num_cclibs = 1;
gc_start_struct.cclib_list = &cclib_struct;
gc_start_struct.cclib_list[0].cclib_name =  "GC_ISDN_LIB";
gc_start_struct.cclib_list[0].cclib_data = parmblk;

if ( gc_Start( &gc_start_struct ) != GC_SUCCESS ) {
    exit(1);
}
gc_util_delete_parm_blk(parmblk);
```

The field extevtdatap of the METAEVENT structure points to EXTENSIONEVT_BLK.

```
typedef struct {
    unsigned char    ext_id;
    GC_PARM_BLK      parmblk;
} EXTENSIONEVTBLK;
```

The following table defines the different possible extension IDs in the GCEV_EXTENSION event.

Event	Description
GCIS_EXEV_NOTIFYGLOBAL when using Springware boards **Note:** When using DM3 boards, this event is not supported.	A DROP request has been received; the request was made by sending the SndMsg_Drop message type via the **gc_Extension(GCIS_EXID_SNDMSG)** function. This event has two different meanings that depend upon the type of call: • Two-party call - the event is a request to disconnect the call. The application should respond by issuing a **gc_DropCall( )**. • Conference call - the event is a request to remove the last party that was added to the conference. The application needs to respond to this request with either a SndMsg_DropAck or SndMsg_DropRej message to indicate the acceptance or rejection of the request. If the request is accepted, the party is dropped from the conference. This event only pertains to a Custom BRI 5ESS switch type.
GCIS_EXEV_CONGESTION when using Springware boards When using DM3 boards, the equivalent event is GCEV_CONGESTION	A CONGESTION message has been received by the application, indicating that the remote end is not ready to accept incoming user information. Use the **gc_GetCallInfo( )** function to retrieve additional information about the event or look into the extension event data.
GCIS_EXEV_DIVERTED when using Springware boards **Note:** When using DM3 boards, this event is not supported.	NAM with divert information has been received by the application. An outgoing call has been successfully diverted to another station.
GCIS_EXEV_DROPACK when using Springware boards **Note:** When using DM3 boards, this event is not supported.	The network has honored a DROP request for a conference call; the request was made by sending the SndMsg_Drop message type via the **gc_Extension(GCIS_EXID_SNDMSG)** function. The event is sent on the corresponding line device. This event pertains only to a Custom BRI 5ESS switch type.
GCIS_EXEV_DROPREJ when using Springware boards **Note:** When using DM3 boards, this event is not supported.	The network has not honored a DROP request for a conference call. The event is sent on the corresponding line device.This event pertains only to a Custom BRI 5ESS switch type.
GCIS_EXEV_FACILITY when using Springware boards When using DM3 boards, the equivalent event is GCEV_FACILITY	A FACILITY REQUEST message has been received by the application.
GCIS_EXEV_FACILITY_ACK when using Springware boards. **Note:** When using DM3 boards, this event is not supported.	A FACILITY_ACKNOWLEDGEMENT message has been received by the application.
GCIS_EXEV_FACILITY_REJ when using Springware boards **Note:** When using DM3 boards, this event is not supported.	A FACILITY_REJECT message has been received by the application.

Event	Description
GCIS_EXEV_FACILITYGLOBAL when using Springware boards **Note:** When using DM3 boards, this event is not supported.	An ISDN_FACILITY message containing a Global CRN value was received. This event is sent on the board level device, as the event is associated with all calls on the device. Upon receipt of this event, the application may issue a **gc_Extension(GCIS_EXID_GETNONCALLMSG)** function to retrieve the data into its local structure or look into the extension event data.
GCIS_EXEV_FACILITYNULL when using Springware boards **Note:** When using DM3 boards, this event is not supported.	An ISDN_FACILITY message was received containing a Dummy (NULL) CRN. Upon receipt of this event, the application may issue a **gc_Extension(GCIS_EXID_GETNONCALLMSG)** function to retrieve the data into its local structure or look into the extension event data.
GCIS_EXEV_INFOGLOBAL when using Springware boards **Note:** When using DM3 boards, this event is not supported.	An ISDN_INFORMATION message containing a Global CRN value was received. This event is sent on the board level device, as the event is associated with all calls on the device. Upon receipt of this event, the application may issue a **gc_Extension(GCIS_EXID_GETNONCALLMSG)** function to retrieve the data into its local structure or look into the extension event data.
GCIS_EXEV_INFONULL when using Springware boards **Note:** When using DM3 boards, this event is not supported.	An ISDN_INFORMATION message was received containing a NULL CRN. Upon receipt of this event, the application may issue a **gc_Extension(GCIS_EXID_GETNONCALLMSG)** function to retrieve the data into its local structure or look into the extension event data.
GCIS_EXEV_L2BFFRFULL when using Springware boards **Note:** When using DM3 boards, this event is not supported.	Reserved for future use.
GCIS_EXEV_L2FRAME when using Springware boards When using DM3 boards, the equivalent event is GCEV_L2FRAME	A data link layer frame has been received by the application. The application should use the **gc_Extension(GCIS_EXID_GETFRAME)** function to retrieve the received frame. It is the application's responsibility to analyze the contents of the frame or look into the extension event data.
GCIS_EXEV_L2NOBFFR when using Springware boards **Note:** When using DM3 boards, this event is not supported.	There are no buffers available to save the incoming frame.
GCIS_EXEV_NOFACILITYBUF when using Springware boards **Note:** When using DM3 boards, this event is not supported.	Facility buffer is not ready.
GCIS_EXEV_NOTIFY when using Springware boards When using DM3 boards, the equivalent event is GCEV_NOTIFY	A NOTIFY message has been received by the application. Use the **gc_GetCallInfo( )** function to retrieve additional information about the event or look into the extension event data.

Event	Description
GCIS_EXEV_NOTIFYGLOBAL when using Springware boards **Note:** When using DM3 boards, this event is not supported.	An ISDN_NOTIFY message containing a Global CRN value was received. This event is sent on the board level device, as the event is associated with all calls on the device. Upon receipt of this event, the application may issue a **gc_Extension(GCIS_EXID_GETNONCALLMSG)** function to retrieve the data into its local structure or look into the extension event data.
GCIS_EXEV_NOTIFYNULL when using Springware boards **Note:** When using DM3 boards, this event is not supported.	An ISDN_NOTIFY message was received containing a Dummy (NULL) CRN. Upon receipt of this event, the application may issue a **gc_Extension(GCIS_EXID_GETNONCALLMSG)** function to retrieve the data into its local structure or look into the extension event data.
GCIS_EXEV_NOUSRINFOBUF when using Springware boards **Note:** When using DM3 boards, this event is not supported.	User IE buffer is not ready.
GCIS_EXEV_NSI when using Springware boards **Note:** When using DM3 boards, this event is not supported.	A Network Specific Indication (NSI) message was received from the network. The application should use **gc_GetCallInfo( )** to retrieve the NSI string(s) or look into the extension event data.
GCIS_EXEV_PLAYTONE when using Springware boards **Note:** When using DM3 boards, this event is not supported.	User-defined tone successfully played.
GCIS_EXEV_PLAYTONEFAIL when using Springware boards **Note:** When using DM3 boards, this event is not supported.	Request to play user-defined tone failed.
GCIS_EXEV_PROGRESSING when using Springware boards When using DM3 boards, the equivalent event is GCEV_PROGRESSING	A PROGRESS message has been received by the application. By default, the firmware will send this event to the application. The application may block this event by clearing the CCMSK_PROGRESS bit. Use the **gc_GetCallInfo( )** function to retrieve additional information about the event or look into the extension event data.
GCIS_EXEV_STATUS when using Springware boards **Note:** When using DM3 boards, this event is not supported.	A STATUS message has been received from the network.
GCIS_EXEV_STATUS_ENQUIRY when using Springware boards **Note:** When using DM3 boards, this event is not supported.	A STATUS_ENQ message has been received from the network.
GCIS_EXEV_STOPTONE when using Springware boards **Note:** When using DM3 boards, this event is not supported.	The tone operation was terminated.
GCIS_EXEV_STOPTONEFAIL when using Springware boards **Note:** When using DM3 boards, this event is not supported.	The request to terminate the playing of a tone failed.

Event	Description
GCIS_EXEV_TIMER when using Springware boards **Note:** When using DM3 boards, this event is not supported.	An unsolicited event indicating that a timer has expired.
GCIS_EXEV_TONEREDEFINE when using Springware boards **Note:** When using DM3 boards, this event is not supported.	The tone(s) in the firmware tone template table were successfully redefined.
GCIS_EXEV_TONEREDEFINEFAIL when using Springware boards **Note:** When using DM3 boards, this event is not supported.	The request to redefine tone(s) in the firmware tone template table failed.
GCIS_EXEV_TRANSFERACK when using Springware boards **Note:** When using DM3 boards, this event is not supported.	A TRANSFER ACKNOWLEDGE message was received from the network. The indicated network has accepted a request to transfer a call.
GCIS_EXEV_TRANSFERREJ when using Springware boards **Note:** When using DM3 boards, this event is not supported.	A TRANSFER REJECT message was received from the network. The indicated network has rejected a request to transfer a call.
GCIS_EXEV_TRANSIT when using Springware boards When using DM3 boards, the equivalent event is GCEV_TRANSIT	After a transfer is established, transit messages are used for relating messages between the originating end and the terminating end.
GCIS_EXEV_USRINFO when using Springware boards When using DM3 boards, the equivalent event is GCEV_USRINFO	A USER INFORMATION message has been received by the application, indicating that a user-to-user information (UUI) event is coming. For example, this event is received in response to a **gc_Extension(GCIS_EXID_SNDMSG)** function call, from the far end, in which the msg_type is SndMsg_UsrInformation. Use the **gc_GetCallInfo( )** function to retrieve the UUI or look into the extension event data. Field parmblk of EXTENSIONEVTBLK will hold following parameters: GCIS_SET_IE, GCIS_PARM_UIEDATA (char array, maximum length can go to MAXLEN_IEDATA): Unprocessed IEs in CCITT format. The IEs are returned as raw data and must be parsed and interpreted by the application.

Update for **Call Rejection Notification for Blocked Channel Recovery**

Because of a new feature in the Service Update, information about recovering blocked channels should be added to **Chapter 4, ISDN-Specific Operations**. Also, the new GCEV_EXTENSION event associated with this feature should be added to **Table 8, Responding to ISDN Events**. For information about this feature, see Section 1.9, “Call Rejection Notification for Blocked Channel Recovery”, on page 30 of this Release Update.

Update for **Dynamic Selection of the Signaling Type for a Trunk**

Because of a new feature in the Service Update, information regarding the dynamic selection of the signaling type for a trunk should be added as a subsection under **Section 4.15, Using Dynamic Trunk Configuration**. Also, in the **ISDN-Specific**

Parameter Reference chapter, a new section for "GCSET_LINE_CONFIG" should be added to include details of the CCPARM_SIGNALING_TYPE parameter ID as described in Section 1.10, "Dynamic Selection of Signaling Type for a Trunk", on page 33 of this Release Update.

Updates to the **Setting the CRC4 Mode for a Trunk** section

Section **4.15.1, Setting the CRC4 Mode for a Trunk** beginning on page 146 applies equally to setting the CRC6 mode. The section should be updated to describe the setting of the CRC mode in general. In addition, the section also applies to setting the ISDN protocol mode (user or network). The revised text for this section is as follows:

4.15.1 Setting the CRC Mode and ISDN Protocol Mode for a Trunk

Note: This feature is only applicable when using DM3 boards.

The **gc_SetConfigData( )** function can be used to change the CRC mode (on or off) and the ISDN protocol mode (user or network) without having to re-download the board firmware. This means that it is not necessary to stop processing calls while the settings are changed on a single trunk.

The **gc_SetConfigData( )** function uses a GC_PARM_BLK structure that contains the configuration information. The GC_PARM_BLK is populated using the **gc_util_insert_parm_val( )** function.

To configure CRC (applicable to E1 systems only), use the **gc_util_insert_parm_val( )** function with the following parameter values:

- **parm_blkpp** = pointer to the address of a valid GC_PARM_BLK structure where the parameter and value are to be inserted
- **setID** = CCSET_LINE_CONFIG
- **parmID** = CCPARM_CRC
- **data_size** = 4 (integer)
- **data** = either 2 (set CRC off) or 3 (set CRC on)

To configure User or Network mode, use the **gc_util_insert_parm_val( )** function with the following parameter values:

- **parm_blkpp** = pointer to the address of a valid GC_PARM_BLK structure where the parameter and value are to be inserted
- **setID** = CCSET_LINE_CONFIG
- **parmID** = CCPARM_USER_NETWORK
- **data_size** = 4 (integer)
- **data** = either 0 (User mode) or 1 (Network mode)

Once the GC_PARM_BLK has been populated with the desired values, the **gc_SetConfigData( )** function can be issued to perform the configuration. The parameter values for the **gc_SetConfigData( )** function are as follows:

- **target_type** = GCTGT_CCLIB_CHAN

- **target_id** = the trunk line device handle, as obtained from **gc_OpenEx( )** with a **devicename** string of ":N_dtiBx:P_ISDN", which can also optionally include a voice device.
- **target_datap** = GC_PARM_BLKP parameter pointer, as constructed by the utility function **gc_util_insert_parm_val( )**
- **time_out** = time interval (in seconds) during which the target object must be updated with the data. If the interval is exceeded, the update request is ignored. This parameter is supported in synchronous mode only, and it is ignored when set to 0.
- **update_cond** = GCUPDATE_IMMEDIATE
- **request_idp** = pointer to the location for storing the request ID
- **mode** = EV_ASYNC for asynchronous execution or EV_SYNC for synchronous execution

Notes: ***1.*** The application must include the *dm3cc_parm.h* header file when using this feature.

2. Call activity must be cleared on the trunk being configured. The application must issue a **gc_ResetLineDev( )** on all devices opened on the trunk before calling **gc_SetConfigData( )**.

3. The configuration changes made by issuing **gc_SetConfigData( )** are not persistent, that is, the CONFIG and FCD files are not updated.

Example

In the following example, assume that **ldev** is a LINEDEV-type variable, properly initialized by a successful call to **gc_OpenEx( )**.

```
GC_PARM_BLKP ParmBlkp = NULL;
long id;
if(DoWeWantToSetCRCInThisSample) {
    int valueCRC = 2;               /* 2 for CRC off, 3 for CRC on */
    gc_util_insert_parm_val(&ParmBlkp, CCSET_LINE_CONFIG, CCPARM_CRC, sizeof(int), valueCRC);
    }
if(DoWeWantToSetModeInThisSample) {
    int valueMode = 0;              /* 0 for User mode, 1 for Network mode */
    gc_util_insert_parm_val(&ParmBlkp, CCSET_LINE_CONFIG, CCPARM_USER_NETWORK, sizeof(int),
                            valueMode);
    }
if(DoWeWantToSetCRCInThisSample || DoWeWantToSetModeInThisSample) {
    gc_SetConfigData(GCTGT_CCLIB_CHAN, ldev, ParmBlkp, 0, GCUPDATE_IMMEDIATE, &id, EV_ASYNC);
    gc_util_delete_parm_blk(ParmBlkp);
    }
if (sr_waitevt(-1) >= 0) {
    METAEVENT meta;
    gc_GetMetaEvent(&meta);
    switch(sr_getevttype()) {
        case GCEV_SETCONFIGDATA:
            printf("Received event GCEV_SETCONFIGDATA(ReqID=%d) on device %s \n",
                  ((GC_RTCM_EVTDATA *)(meta.evtdatap))->request_ID, ATDV_NAMEP(sr_getevtdev()));
            break;
        case GCEV_SETCONFIGDATA_FAIL
            printf("Received event GCEV_SETCONFIGDATAFAIL(ReqID=%d) on device %s, Error=%s\n",
                  ((GC_RTCM_EVTDATA *)(meta.evtdatap))->request_ID, ATDV_NAMEP(sr_getevtdev()),
                  ((GC_RTCM_EVTDATA *)(meta.evtdatap))->additional_msg);
            break;
        default:
            printf("Received event 0x%x on device %s\n", sr_getevttype(),
                  ATDV_NAMEP(sr_getevtdev()));
```

```
                break;
        }

}
```

Update to the **ISDN Network Firmware** section

Because of a feature introduced in the Service Update, ISDN network-side mode is now supported and fully qualified for operation in a deployment environment. In **Section 7.2, ISDN Network Firmware**, the following note at the beginning of the section is no longer applicable:

Note: With the exception of the Q.Sig protocol on DM3 boards, where the User-side and Network-side protocols are symmetrical, the Network-side firmware for all ISDN protocols (for both DM3 and Springware boards) is provided for back-to-back testing purposes only and is not fully qualified for operation in a deployment environment.

For further information about this feature, see Section 1.13, "ISDN Network Side Conformance to Network Protocol Standards ITU-T Q921 and Q931", on page 42 of this Release Update.

3.4.12 Global Call SS7 Technology Guide

Update to **Section 2.1, Using Global Call with SS7**

SS7HDCN16 is also supported in DTI (clear channel mode). See Section 1.2, "Global Call Support for Time Slots on SS7 Boards Running in DTI Mode", on page 18 of this Release Update.

Update to **Section 2.1.1, SS7 Interface Boards**

The "Note" is replaced by the following: "Note: Multiple Intel NetStructure SS7 boards are now supported in a system. However, only one Intel NetStructure SS7 board can be used for processing SS7 signaling links in a system. The rest of the boards must be configured in clear channel mode (DTI mode)."

Update to **Section 3.3, Global Call Software Configuration (gcss7.cfg)**

Subsection **config.txt Related Parameters** should be renamed to **config.txt Related Parameters for Clear Channel for ISUP** and the information in Section 1.2.3.1, "Global Call SS7 Software Configuration (gcss7.cfg)", on page 18 of this Release Update should be added to that section.

Update to **Chapter 5, SS7-Specific Operations**

Because of a new feature, **Alarm Handling for SS7 Boards,** in the Service Update, this chapter should include a section on alarm handling for SS7 boards. See the Section 1.4.3.2, "Alarm Handling for SS7 Boards", on page 24 of this Release Update.

Update to **Section 8.1, Global Call Functions Supported by SS7**

Because of a new feature, **Alarm Handling for SS7 Boards,** in the Service Update, this chapter should indicate that the GCAMS functions (with the exception of **gc_TransmitAlarms( )** and **gc_StopTransmitAlarms( )**) are supported.

Update to **Chapter 10, SS7-Specific Event Cause Codes**

Because of a new feature, **Alarm Handling for SS7 Boards,** in the Service Update, this chapter should include the layer 1 alarm related event cause codes. See the Section 1.4.3.4, "SS7-Specific Event Cause Codes for Layer 1 Alarms", on page 26 of this Release Update.

3.4.13 IP Media Library API Programming Guide

Update to **Section 6.1, Introduction to DTMF Handling**

The fourth paragraph in **Section 6.1, Introduction to DTMF Handling** (page 21) and the note that follows the fourth paragraph should be ignored. The IPM_RFC2833MUTE_AUDIO parameter that the paragraph refers to is not supported; DTMF audio is always muted when in RFC2833 mode. Similarly, Step 5 in the procedure in **Section 6.2.3, Setting RFC 2833 Mode** (page 24) should also be ignored. (PTR# 33826)

Updates to **Section 7.2, QoS Alarm Types and Thresholds**

The list item for **EVT_DTMFDISCARDED** (page 31) should be ignored because this alarm is not supported.

The list item for **EVT_LOSTPACKETS** (page 31) should correctly refer to QOSTYPE_LOSTPACKETS, and should note that the alarm is only supported for IPT Series boards.

The list item for **EVT_JITTER** (page 32) should correctly refer to QOSTYPE_JITTER.

The list item for **EVT_ROUNDTRIPLATENCY** (page 32) should correctly refer to QOSTYPE_ROUNDTRIPLATENCY.

Update to **Section 7.4, Using QoS Alarms**

The example code is missing the declaration and initialization for the **l_pVoid** variable within the **CheckEvent( )** subroutine on page 35. The code should include the line:

```
void* l_pVoid = sr_getevtdatap();
```

3.4.14 IP Media Library API Library Reference

Update to **ipm_DisableEvents( )**

On the reference pages for **ipm_DisableEvents( )** (page 18), the description of the EVT_LOSTPACKETS value for the **pEvents** parameter should indicate that this QoS alarm is only supported for IPT Series boards.

Update to **ipm_EnableEvents( )**

On the reference pages for **ipm_EnableEvents( )** (page 22), the description of the EVT_LOSTPACKETS value for the **pEvents** parameter should indicate that this QoS alarm is only supported for IPT Series boards.

Update to **IPM_QOS_ALARM_DATA** data structure

On the reference page for the **IPM_QOS_ALARM_DATA** data structure (page 114), the description of the QOSTYPE_LOSTPACKETS value for the **eQoSType** field should indicate that this alarm is only supported for IPT Series boards.

Update to **IPM_QOS_SESSION_INFO** data structure

On the reference page for the **IPM_QOS_SESSION_INFO** data structure (page 116), the description of the QOSTYPE_LOSTPACKETS value for the **eQoSType** field should indicate that this alarm is only supported for IPT Series boards.

Update to **IPM_QOS_THRESHOLD_DATA** data structure

On the reference page for the **IPM_QOS_THRESHOLD_DATA** data structure (pages 117 and 118), the description of the QOSTYPE_LOSTPACKETS value for the **eQoSType** field and the subentry for QOSTYPE_LOSTPACKETS in the description of

the **unFaultThreshold** field should indicate that this alarm is only supported for IPT Series boards. In Table 6, Quality of Service Parameter Defaults for DM/IP Series Boards, all cells in the “Lost Packets” row should indicate “n/a”.

3.4.15 Modular Station Interface API Programming Guide

Update to **Chapter 4, Application Development Guidelines**

Add a note to Chapter 4 stating that when using HDSI boards, application developers **must** follow the recommendation outlined in “Performance Considerations” in Chapter 2 of the *Standard Runtime Library API Programming Guide*.

3.4.16 Modular Station Interface API Library Reference

Update to **ms_SendData( )**

The following information is added to the description of the **ms_SendData( )** function:

Make sure an interval of at least 2 seconds elapses between the reception of an MSEV_RING event with MSMM_RNGOFFHK event data and a call to the **ms_SendData**( ) function to ensure reliable reception of call waiting caller ID. If there is still a problem receiving the call waiting caller ID, it may be due to the configuration of the phone.

Update to MS_CDT data structure

A note should be added to the MS_CDT chan_sel field indicating that MSPN_STATION is supported on Springware products only and MSPN_TS is supported on DI, HDSI, and Springware products. (PTR# 35565)

3.4.17 Porting Global Call H.323 Applications from Embedded Stack to Host-Based Stack Application Note

There are currently no updates to this document.

3.4.18 Standard Runtime Library API Programming Guide

There are currently no updates to this document.

3.4.19 Standard Runtime Library API Library Reference

There are currently no updates to this document.

3.4.20 Voice API Programming Guide

Information on new features for Intel NetStructure® DM/V and DM/V-A Boards, 8K linear coder and ETSI-FSK (SMS) parameters, is provided here

Update to the **Recording and Playback** chapter

The following information should be added in Table 9, Voice Encoding Methods (DM3 Boards) of Section 8.5 Voice Encoding Methods, in Chapter 8, Recording and Playback.

Voice Encoding Methods (DM3 Boards)

Digitizing Method	Sampling Rate (kHz)	Resolution (Bits)	Bit Rate (kbps)	File Format
Linear PCM	8	8	64	VOX, WAVE
Linear PCM	8	16	128	VOX, WAVE

Update to **Section 8.7, Transaction Record** (IPY00006537 = PTR# 35666)

The following paragraph should be added to this section:

For information on running transaction record on a single board, see the technical note posted on the Intel Telecom Support Resources web site at:
http://resource.intel.com/telecom/support/tnotes/gentnote/dl_soft/tn253.htm

Update to the **Send and Receive FSK Data** chapter

The following information should be added as a new section following section 10.5 Two-Way ADSI in Chapter 10, Send and Receive FSK Data.

Fixed-Line Short Message Service (SMS)

Fixed-line short message service or SMS is a service that allows text messages to be sent and received in the PSTN network. SMS is also known as small message service or text messaging. SMS is not supported on Springware boards.

The voice library supports the creation of fixed-line SMS applications through the **dx_RxIottData( )**, **dx_TxIottData( )**, and **dx_TxRxIottData( )** functions.

Fixed-line SMS solutions can be done in two ways: using the standard Bellcore ADSI specification or using the ETSI-FSK specification ETSI ES 201 912.

The ETSI-FSK specification differs from the Bellcore ADSI FSK specification in these ways:

- It uses a different physical layer. Settings for channel seizure and mark length differ. For more information on FSK transmission requirements, see ITU-T EN 300 659-2 specification.
- It uses different handshaking and timing specifications.

To set the voice channel to ETSI compatibility, use the FSK parameters: DXCH_FSKSTANDARD, DXCH_FSKCHSEIZURE, and DXCH_FSKMARKLENGTH.

Update to the **Send and Receive FSK Data** chapter

The following new ETSI-FSK parameters should be included in section 10.6.1 Support on DM3 boards in Chapter 10, Send and Receive FSK Data.

Support on DM3 Boards

DXCH_FSKSTANDARD channel parameter

Specifies the FSK protocol standard, which is used for transmission and reception of FSK data. Using this channel parameter, the protocol standard can be set to either DX_FSKSTDBELLCORE (Bellcore standard) or DX_FSKSTDETSI (ETSI standard). The default value is DX_FSKSTDBELLCORE.

If you set DXCH_FSKSTANDARD to DX_FSKSTDETSI, it is recommended that you explicitly specify values for the DXCH_FSKCHSEIZURE and DXCH_FSKMARKLENGTH parameters.

DXCH_FSKCHSEIZURE channel parameter

For a given FSK protocol standard specified in DXCH_FSKSTANDARD, this parameter allows the application to set the channel seizure.

When *transmitting* data, the range of possible values is 0 to 300 bits. If you specify a value outside of this range, the library uses 300 bits as the default when transmitting data. If you do not specify a value for channel seizure, the library uses 0 bits as the default.

When *receiving* data, the range of possible values is 0 to 60 bits. If you specify a value outside of this range, it uses 60 bits as the default when receiving data. If you do not specify a value for channel seizure, the library uses 0 bits as the default.

DXCH_FSKMARKLENGTH channel parameter

For a given FSK protocol standard specified in DXCH_FSKSTANDARD, the DXCH_FSKMARKLENGTH parameter allows the application to set the mark length.

When *transmitting* data, the range of possible values is 80 to 180 bits. If you specify a value outside of this range, the library uses 180 bits as the default when transmitting data. If you do not specify a value for mark length, the library uses 80 bits as the default.

When *receiving* data, the range of possible values is 0 to 60 bits. If you specify a value outside of this range, it uses 30 bits as the default when receiving data. If you do not specify a value for mark length, the library uses 0 bits as the default.

Update to **Section 13.1.9, Guidelines for Creating User-Defined Tones** (IPY00006580 = PTR# 34546)

The following guideline should be added to this section:

- On DM3 boards, building and adding tones of zero frequency values to a tone template can cause firmware failures.

Update to **Chapter 17, Building Applications**

Run-time linking using the source code in the CLIB subdirectory is no longer supported. Run-time linking can be accomplished using Windows functions. In the Voice API Programming Guide, **Section 17.2.3, Run-time Linking**, should be revised as follows (PTR# 32966):

Run-time linking resolves the entry points to the Intel DLLs when the application is loaded and executed. This allows the application to contain function calls that are not contained in the DLL that resides on the target system.

To use run-time linking, the application can call the Windows **LoadLibrary( )** function to load a specific technology DLL and a series of **GetProcAddress( )** function calls to set up the address pointers for the functions.

Functions not supported

The **r2_creatfsig( )** and **r2_playbsig( )** functions, which were previously provided for backward compatibility only, are no longer supported. All references to these functions should be deleted. R2MF signaling is typically accomplished through the Dialogic® Global Call API.

3.4.21 Voice API Library Reference

Updates for new features for Intel NetStructure® DM/V and DM/V-A Boards, 8K linear coder and ETSI-FSK (SMS) parameters

Play and record functions and the DX_XPB data structure support 8K linear coder: 8kHz with 8-bit samples (64 kbps) and 8kHz with 16 bit samples (128 kbps).

The **dx_setparm( )** function includes support for the following new ETSI-FSK parameters:

- DXCH_FSKSTANDARD channel parameter
- DXCH_FSKCHSEIZURE channel parameter
- DXCH_FSKMARKLENGTH channel parameter

For more information on these parameters, see Section 3.4.20, “Voice API Programming Guide”, on page 159 of this Release Update.

Update to FEATURE_TABLE data structure

With the Service Update, the FT_CALLERID bit is now being used to indicate FSK support on DM3 boards.

Functions not supported

The **r2_creatfsig( )** and **r2_playbsig( )** functions, which were previously provided for backward compatibility only, are no longer supported. All references to these functions should be deleted. R2MF signaling is typically accomplished through the Dialogic® Global Call API.

3.5 Demonstration Software Documentation

This section contains updates to the following documents (click the title to jump to the corresponding section):

- Continuous Speech Processing Demo Guide
- Global Call API Demo Guide
- High Availability for Windows Demo Guide
- IP Gateway (Global Call) Object Oriented Demo Guide
- IP Media Server (Global Call) Demo Guide for Windows
- IP Media Gateway (IPML) Demo Guide
- IP Multicast Client (IPML) Demo Guide

- IP Multicast Server (IPML) Demo Guide

3.5.1 Continuous Speech Processing Demo Guide

There are currently no updates to this document.

3.5.2 Global Call API Demo Guide

There are currently no updates to this document.

3.5.3 High Availability for Windows Demo Guide

There are currently no updates to this document.

3.5.4 IP Gateway (Global Call) Object Oriented Demo Guide

There are currently no updates to this document.

3.5.5 IP Media Server (Global Call) Demo Guide for Windows

There are currently no updates to this document.

3.5.6 IP Media Gateway (IPML) Demo Guide

There are currently no updates to this document.

3.5.7 IP Multicast Client (IPML) Demo Guide

Update to the **Hardware Requirements** section

The following note is added to **Section 2.1**, **Hardware Requirements** (PTR# 31488):

Note: When using a single span DM/IP board, the demo supports only one board in the system.

3.5.8 IP Multicast Server (IPML) Demo Guide

Update to the **Hardware Requirements** section

The following note is added to **Section 2.1**, **Hardware Requirements** (PTR# 31488):

Note: When using a single span DM/IP board, the demo supports only one board in the system.

3.6 Online Help

This section contains updates to the following documents (click the title to jump to the corresponding section):

- DCM Online Help
- Install Help

3.6.1 DCM Online Help

Updates for the **Trunk Configuration Property Sheet**

Because of a feature introduced in the Service Update, you can now mix ISDN trunks and clear channel trunks on the same DMN160TEC or DMT160TEC board using the Trunk Configuration property sheet of the configuration manager (DCM). Previously, you had to edit the CONFIG file to accomplish this. For information about the changes to the Trunk Configuration property sheet, see Section 1.19, “Mixing ISDN and Clear Channel on a DMN160TEC or DMT160TEC Board”, on page 51 of this Release Update.

Because of a feature introduced in the Service Update, you can now change the signaling type at runtime of any trunk from ISDN to clear channel or vice versa on a DMN160TEC or DMT160TEC board for E1 configurations only. In order to enable this capability, a new parameter, **Dynamic_ISDN_CC**, has been added to the Trunk Configuration property sheet. For information about this feature and the new parameter, see Section 1.10, “Dynamic Selection of Signaling Type for a Trunk”, on page 33 of this Release Update.

Update to **PciID** parameter help topic (Physical property sheet)

The information in the help topic for the **PciID** parameter should read as follows:

Description: A positive integer or hexadecimal value in which the lower 5 bits specify a board’s rotary-switch setting (PCI boards) or the physical slot number location of the board (CompactPCI boards). The rotary-switch setting for PCI boards can be the same for all PCI boards in the system if it is set to 0.

Note: The PciID parameter is set by the system software and should not be changed by the user.

Update to **PhysicalSlotNumber** parameter help topic (Physical property sheet)

The information in the help topic for the **PhysicalSlotNumber** parameter should read as follows:

Description: For a PCI board, specifies the board’s rotary-switch setting. The rotary-switch setting for DM3 architecture PCI boards can be the same for all boards in the system if the value is set to 0.

For a CompactPCI board, specifies the number of the physical slot in which the board is installed. A value of 1 indicates the first slot in the chassis. (The chassis slot numbers are usually marked on the front of the chassis.)

Settings: For a PCI board, 0 to 15

For a CompactPCI board, a positive integer or hexadecimal value.

Note: This parameter is read-only and cannot be modified through the DCM.

3.6.2 Install Help

There are currently no updates to this document.