Global Call SS7 Technology User's Guide

for Windows Operating Systems

05-1380-006

Publication Date: January 2003

Intel Converged Communications
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 Communications Systems Products website at:
http://www.intel.com/network/csp/

For **Sales Offices** and other contact information, visit the Intel Telecom Building Blocks Sales Offices page at:
http://www.intel.com/network/csp/sales/

Table of Contents

List of Figures

List of Tables

Preface

Purpose of this Guide

This guide is for users who use the Global Call Application Programming Interface (API) to develop applications in a Signaling System 7 (SS7) environment.

This guide is not intended to be used alone, but should be used in conjunction with the *Global Call API Library Reference*.

Products Covered By This Guide

This guide describes how to use the Global Call API to develop applications that use the following products to connect to an SS7 network:

- Intel® NetStructure™ SS7 ISA (PCCS6) boards
- Intel® NetStructure™ SS7 PCI boards
- Intel® NetStructure™ SS7 CompactPCI boards

  NOTE: Intel® NetStructure™ SS7 ISA (PCCS6), PCI and CompactPCI boards are single board SS7 solutions.
- DSC131 Intel® NetStructure™ Signaling Interface Unit (SIU)
- DSC231 Intel® NetStructure™ Signaling Interface Unit (SIU)
- DSC520 Intel® NetStructure™ Signaling Interface Unit (SIU)

  NOTE: The DSC131, DSC231 and DSC520 SIUs are SS7 server solutions.

The various supported configurations are described in Chapter 2, “Global Call SS7”. For more information about these products, see the documentation supplied with the products.

Organization of This Guide

This guide provides information for developing SS7 applications and details specific usage of Global Call functions in SS7 applications as follows:

Chapter 1 - Provides an overview of SS7 and its use in computer telephony applications.

Chapter 2 - Describes the use of Global Call SS7 in an SS7 environment and the overall architecture of a Global Call SS7 system.

Chapter 3 - Describes how to configure the Datakinetics environment and the Intel® Dialogic system software for use with Global Call SS7.

Chapter 4 - Provides the additional functionality of specific Global Call API functions when used in SS7 applications, as well as any specific restrictions.

Chapter 5 - Describes how to use Global Call to perform certain SS7-specific tasks.

Chapter 6 - Describes how to troubleshoot the Global Call SS7 system.

Chapter 7 - Describes the data structures and associated parameters used by Global Call SS7.

Appendix A - Explains SS7-specific error codes.

Appendix B - Provides some scenarios for call setup and call release.

Appendix C - Includes sample *system.txt* and *config.txt* files for several hardware configurations.

Related Publications

The following is a list of related documentation:

- *DSC131/DSC231 User's Manual*
- *SS7 Programmer's Manual for PCCS6*

- *SS7 Programmer's Manual for SPCI4, SPCI2S and CPM8*
- *System7 ISUP Programmer's Manual*
- *System7 TUP Programmer's Manual*
- *System7 Software Environment Programmer's Manual*
- *Intel® NetStructure™ SIU131/231 User's Manual*
- *Intel® NetStructure™ SIU520 Developer's Manual*
- *Intel® NetStructure™ SIU520 Migration Guide*
- *Global Call API Library Reference*
- *Global Call API Programming Guide*

NOTE: The SS7 stack and system documentation is available for download at *http://www.intel.com/network/csp/trans/datakinetics.htm*

1. SS7 Overview

1.1. SS7 and Computer Telephony

Signaling System 7 (SS7) is a common-channel signaling (CCS) system that defines the procedures and protocol by which network elements (signaling points) in the public switched telephone network (PSTN) exchange information over a digital signaling network to facilitate wireline and wireless (cellular) call setup, routing and control.

In an SS7 network, control messages (packets) are routed through the network to perform call management (setup, maintenance, and termination) and network management functions. Therefore, the common-channeling signaling SS7 network is a packet-switched network, even though the network being controlled can be a circuit-switched network (PSTN).

An SS7 network is comprised of network elements connected together using signaling links. Such a network element that is capable of handling SS7 control messages is called a **Signaling Point** (SP). All signaling points in a SS7 network are identified by a unique code known as a point code.

There are three different basic types of network elements:

- **Signaling Transfer Point** (STP) - A signaling point that is capable of routing control messages; that is, a message received on one signaling link is transferred to another link.
- **Service Control Point** (SCP) - Contains centralized network databases for providing enhanced services. An SCP accepts queries from an SP and returns the requested information to the originator of the query. For example, when an 800 call is initiated by a user, the originating SP sends a query to an 800 database (at the SCP) requesting information on how to route the call. The SCP returns the routing information to the SP originating the query and the call proceeds.
- **Service Switching Point** (SSP) - A signaling point in a switching office, either a local exchange or a tandem office. An SSP has the capability to control voice

circuits via a voice switch. The SSP can either integrate the voice switch or can be an adjunct computer to the voice switch.

Network elements are interconnected using signaling links. A **signaling link** is a bidirectional transmission path for signaling, comprised of two data channels operating together in opposite directions at the same data rate. The standard rate on a digital transmission channel is 56 or 64 kilobits per second (kbps), although the minimum signaling rate for call control applications is 4.8 kbps. Network management applications may use bit rates lower than 4.8 kbps.

Figure 1 shows an example of an SS7 network that carries signaling information for the underlying PSTN network nodes.

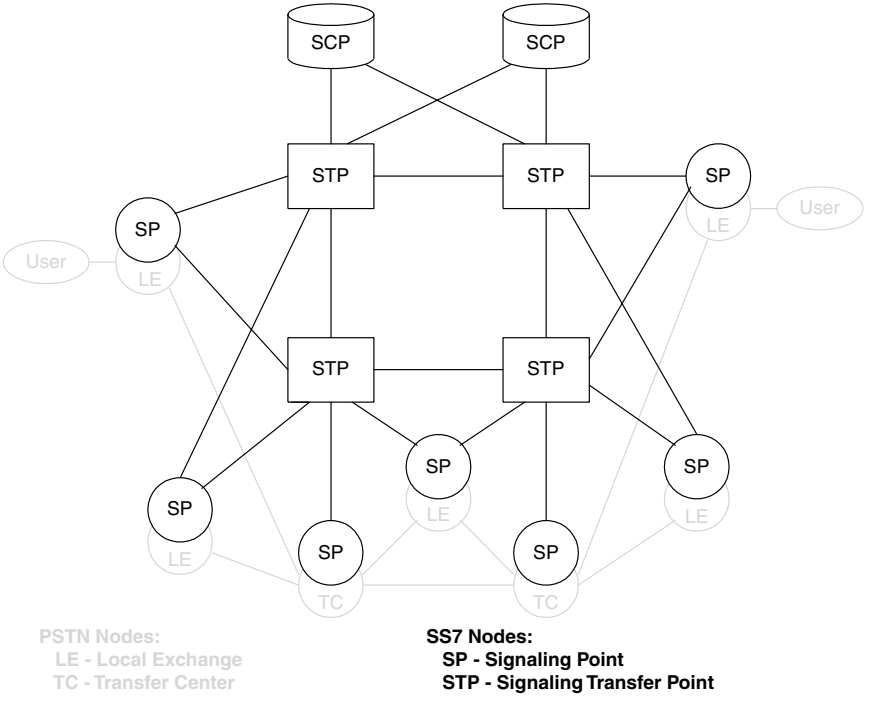


Figure 1. Signaling and Information Transfer Networks

The signaling network is independent of the circuit-switched network. Signaling links can be physically located on trunks that carry voice circuits, but can also be completely independent, or even use a different transmission medium (for example, serial V.35). SSPs are the bridges between both networks.

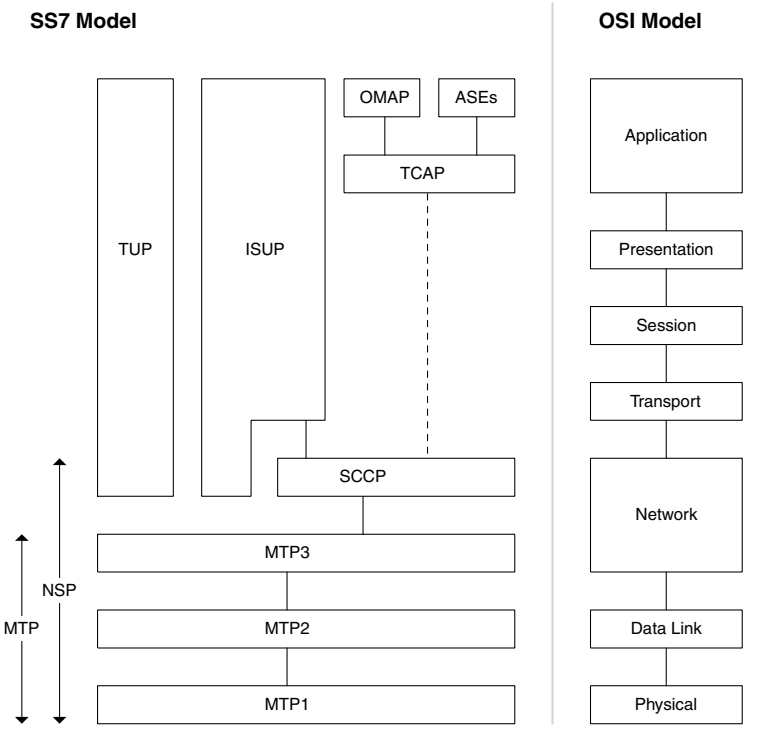
To ensure reliable transfer of signaling information in an environment susceptible to transmission disturbances or network failures, an SS7 network employs error detection and error correction on each signaling link. An SS7 network is normally designed with redundant signaling links and includes functions for the automatic diversion of signaling traffic to alternative paths in case of link failures.

Another type of network element that appears in an Intelligent Network (IN) is the Intelligent Peripheral (IP). An IN is a service-independent telecommunications network, that is, a network in which intelligence is taken out of the switch and placed in computer nodes that are distributed throughout the network. An IP is an SP that provides enhanced services to the SSP, usually under control of an SCP. Those services range from providing user-input prompts and collecting digits to providing a complete service application.

1.2. SS7 Protocol Stack

The hardware and software functions of the SS7 protocol are divided into functional abstractions called levels. These levels map loosely to the Open Systems Interconnect (OSI) 7-layer reference model defined by the International Standards Organization (ISO). This model describes the structure for modeling the interconnection and exchange of information between users in a communications system.

Figure 2 shows the layers of the SS7 protocol stack when transporting SS7 signaling over the PSTN and how the layers relate to the layers of the OSI Model.



Legend:
OMAP - Operations Maintenance Application Part
ASEs - Application Service Elements
TCAP - Transaction Capabilities Application Part
ISUP - ISDN User Part
TUP - Telephony User Part
SCCP - Signaling Connection Control Part
MTP - Message Transfer Part
NSP - Network Service Part

Figure 2. SS7 Protocol Stack Layers

1.2.1. Lower Stack Layers for SS7 Over a Circuit-Switched Network

When transporting SS7 signaling over a circuit-switched network, the lowest three levels of the SS7 stack, called the **Message Transfer Part** (MTP), provide a

reliable but connectionless (datagram or packet style) service for routing messages through the SS7 network. This service is used by the various user parts described in Section 1.2.2, “Upper Stack Layers”, on page 5.

The MTP is subdivided into three parts as follows:

- **MTP1**, also called the **signaling data link** layer, is concerned with the physical and electrical characteristics of the signaling links. MTP1 corresponds to the physical layer of the OSI model.
- **MTP2**, also called the **signaling link** layer, is a data link control protocol that provides for the reliable sequenced delivery of data across a signaling data link. MTP2 corresponds to the data link layer of the OSI model.
- **MTP3**, also called the **signaling network** layer, provides for routing data across multiple STPs from control source to control destination. MTP3 corresponds to a part of the network layer of the OSI model.

The connectionless nature of the MTP provides a low-overhead facility tailored to the requirements of telephony. However, the MTP does not provide all the services of the corresponding OSI Network layer. To support Integrated Services Digital Network (ISDN) applications such as network management that requires expanded addressing capability and reliable message transfer, a separate module is provided:

- **Signaling Connection Control Part** (SCCP), defines a wide variety of network-layer services. SCCP corresponds to part of the network layer of the OSI model.

The MTP and the SCCP together form the **Network Service Part** (NSP). The resulting split in OSI network functions between MTP and SCCP has the advantage that the higher-overhead SCCP services can be used only when required, and the more efficient MTP services can be used in other applications.

1.2.2. Upper Stack Layers

The upper parts of the SS7 protocol stack are concerned with the actual contents of the SS7 messages and are sometimes called application layers. These include:

- **ISDN User Part** (ISUP), provides the signaling needed for basic ISDN circuit-mode bearer services as well as ISDN supplementary services having end-to-end significance. ISUP is the protocol that supports ISDN in the Public

Switched Telephone Network. It corresponds to the transport, session, presentation, application layers and part of the network layer of the OSI model.

- **Telephony User Part** (TUP), an ISUP predecessor in providing telephony signaling functions. TUP has now been made obsolete by ISUP in most countries and in the international network. The TUP corresponds to the transport, session, presentation, application layers and part of the network layer of the OSI model.
- **Transaction Capabilities Application Part** (TCAP), provides the mechanisms for transaction-oriented (rather than connection-oriented) applications and functions. The TCAP corresponds to the application layer in the OSI model. TCAP is often used for database access by the SS7 switches but has many other applications through the network.
- **Operations and Maintenance Application Part** (OMAP), specifies network management functions and messages related to operations and maintenance. The OMAP corresponds to the application layer in the OSI model.
- **Application Service Elements** (ASEs), represent user parts that are highly application-specific, for example:
  - **Intelligent Network Application Part** (INAP)
  - **Mobile Application Part** (MAP), provides the signaling functions necessary for the mobile capabilities of voice and non-voice applications in a mobile network
  - **IS41**, an ANSI signaling standard used in cellular networks

For any application, all three MTP layers and at least one application layer are required. Typically, the word "user" in modules such as ISUP, TUP and so on explicitly identifies the module as a user of the transport mechanism MTP.

SS7 computer telephony applications that transport SS7 signaling over a circuit-switched network can use the ISUP (on top of the MTP layers) to control voice circuits, and sometimes TCAP to query for information or to receive commands from a SCP.

1.3. For More Information on SS7

The following publications provide information about SS7 fundamentals:

- Common-Channel Signaling, Richard J. Manterfield, IEEE Telecommunications Series 26
  1991, Peter Peregrinus Ltd. on behalf of the IEEE
  ISBN 0 86341 240 8
- Signaling System #7, Travis Russel
  1995, McGraw-Hill
  ISBN 0-07-054991-5
- ISDN - Concepts, Facilities and Services, Gary C. Kessler, Peter V. Southwick, (Chapter 10)
  1997, McGraw-Hill, 3rd Edition
  ISBN 0-07-034249-0
- ISDN & SS7 - Architectures for Digital Signaling Networks, Uyless Black
  1997, Prentice Hall
  ISBN 0-13-259193-6
- High-Speed Networks: TCP/IP and ATM Design Principles, William Stallings
  1997, Prentice Hall
  ISBN: 0135259657
- SS7 Basics, Toni Beninger
  1991, Telephony Division of Intertec Publishing Corp.

The following web sites provide background information on SS7 fundamentals when SS7 signaling is used over a circuit-switched network:

- Microlegend SS7 Tutorial - *http://www.pt.com/tutorials/ss7/*
- Web ProForums - *http://www.iec.org/online/tutorials/ss7/*
- CellStream SS7 online tutorial - *http://www.cellstream.com/prod01.htm*

The following web site provides SS7 support:

- DataKinetics home page - *http://www.intel.com/network/csp/trans/datakinetics.htm*

All URLs and site content were verified at the time of writing.

2. Global Call SS7

2.1. Using Global Call with SS7

The SS7 signaling system is a packet-switched data network that forms the backbone of the international telecommunications network. SS7 plays an important role in both wireline and wireless networks. SS7 provides two basic types of services:

- **Call Control** - SS7 provides fast and reliable common channel or out-of-band signaling for call control. At the heart of the SS7 call control function is a network of highly-reliable packet switches called Signal Transfer Points (STPs).
- **Intelligent Network** - The SS7 network enables the implementation of Intelligent Network (IN) and Advanced Intelligent Network (AIN) services. SS7 messages traverse STPs and enlist the use of System Control Points (SCPs), Service Switching Points (SSPs) and Intelligent Peripherals to deliver these services to the user.

Global Call provides a common call control interface for applications, regardless of the signaling protocol needed to connect to the local telephone network. This manual describes the use of Global Call to perform call control functions in a network that supports SS7 signaling.

For SS7 and other protocols, Global Call provides a higher level of abstraction for call control, shielding application developers from the need to deal with the low-level details.

NOTE: Global Call covers only the call control aspects of SS7. It does not provide an API for other user parts such as TCAP and INAP.

Currently, Global Call SS7 supports the ISUP protocol (ANSI version T1.609 and ITU versions Q.761 to Q.764) and the TUP protocol.

Global Call supports the SS7 solutions implemented using Intel® NetStructure™ SS7 hardware and software. Solutions are based on the following hardware and software components:

- SS7 Interface Boards: Intel® NetStructure™ SS7 ISA (PCCS6), PCI, and CompactPCI boards
- Signaling Unit Interface (SIU)
- Intel® NetStructure™ SS7 Protocols

NOTE: The Intel® NetStructure™ SS7 ISA (PCCS6) boards are licensed to handle either 64 or 256 circuits. The SIUs can be licensed to handle up to 4,096 circuits and 16,384 circuits for the DSC131 and DSC231 boards respectively. Contact Intel for information about licensing.

2.1.1. SS7 Interface Boards

SS7 interface boards (Intel® NetStructure™ SS7 boards) are intelligent SS7 signaling boards for use in PC-compatible computers. Intel® NetStructure™ SS7 boards combine on-board support for the SS7 common channel signaling protocols, one, two, four or eight interfaces depending on the board type, and CT Bus local PCM time slots on a mezzanine bus. A dedicated on-board processor ensures that performance is independent of the load on the host PC. Downloadable operating software makes the board easy to upgrade when protocol specification changes are necessary.

The Intel® NetStructure™ SS7 ISA (PCCS6) board is available with one or two E-1 or T-1 line interfaces, and because SS7 signaling is carried separately from the PCM stream in some situations, a V.35-compatible serial interface is also provided. The SCbus connection allows system integration with the complete set of Intel® Dialogic solutions and a wide range of third-party voice, data, and fax products. A digital cross-connect switch allows voice and signaling channels to be connected between the line interfaces, the SCbus time slots, and the protocol processor. Only one Intel® NetStructure™ SS7 ISA (PCCS6) board can be used in a system.

The Intel® NetStructure™ SS7 PCI board comes with four E-1/T-1 interfaces and an H.100 PCM Highway.

The Intel® NetStructure™ SS7 CompactPCI board provides up to eight E-1/T-1 interfaces, V.11 (V.35-compatible) serial ports, and an H.100 PCM Highway.

Figure 3, Figure 4, and Figure 5 show some configurations that use the Intel® NetStructure™ SS7 ISA (PCCS6) board in conjunction with Intel® Dialogic boards in a single chassis that in each case supports up to 256 ports. Table 1 summarizes the features and benefits of each configuration.

Table 1. Intel® NetStructure™ SS7 ISA (PCCS6) Board Configurations - Features and Benefits

Configuration	Features	Benefits
Intel® NetStructure™ SS7 ISA Board Configuration 1	• T-1/E-1 line with SS7 signaling connected to the Intel® NetStructure™ SS7 ISA board • Voice channels routed through the Intel® NetStructure™ SS7 ISA board via the SCbus • SS7 T-1/E-1 managed by the Intel® NetStructure™ SS7 ISA board	• Multiple signaling reliability with up to three signaling links
Intel® NetStructure™ SS7 ISA Board Configuration 2	• SS7 link and bearer channels enter through Intel® Dialogic network interface board • T-1/E-1 with SS7 Signaling channel connects to a voice board • The SS7 signaling is routed to the Intel® NetStructure™ SS7 ISA board via the SCbus	• SCbus bus clocking managed via Intel® Dialogic boards • All voice and data resources managed by Intel® Dialogic boards

Table 1. Intel® NetStructure™ SS7 ISA (PCCS6) Board Configurations - Features and Benefits (Continued) (Continued)

Configuration	Features	Benefits
Intel® NetStructure™ SS7 ISA Board Configuration 3	• The SS7 link is connected via a synchronous V.35 connection • All T-1/E-1 trunks (bearing voice circuits) enter through Intel® Dialogic network interface boards	• Separates the signaling channel from the bandwidth channels • All signaling controlled using V.35 clocking via two V.11 connections on the Intel® NetStructure™ SS7 ISA board

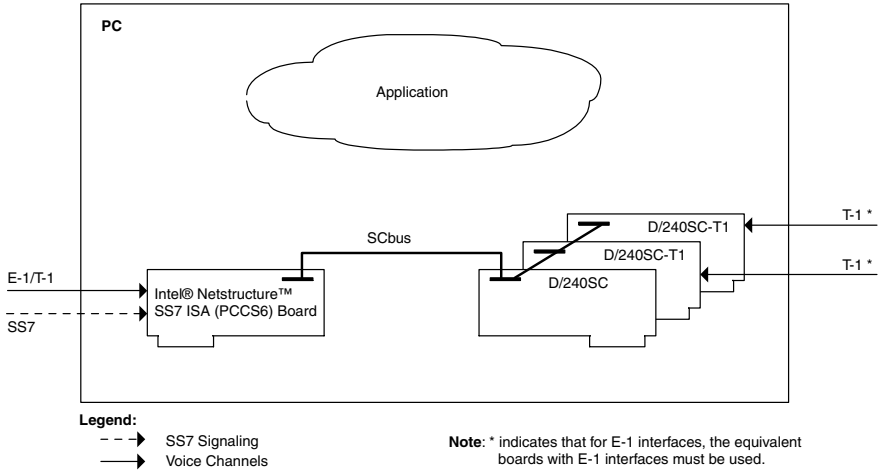


Figure 3. Intel® NetStructure™ SS7 ISA (PCCS6) Board Configuration 1

The key features in this configuration are:

- The T-1/E-1 line with the SS7 signaling is connected to the Intel® NetStructure™ SS7 ISA (PCCS6) board

- B-channels are routed through the Intel® NetStructure™ SS7 ISA (PCCS6) board to voice resource via SCbus
- The SS7 T-1/E-1 is managed by the Intel® NetStructure™ SS7 ISA (PCCS6) board
- Other T-1/E-1 trunks are managed by Intel® Dialogic network interface boards

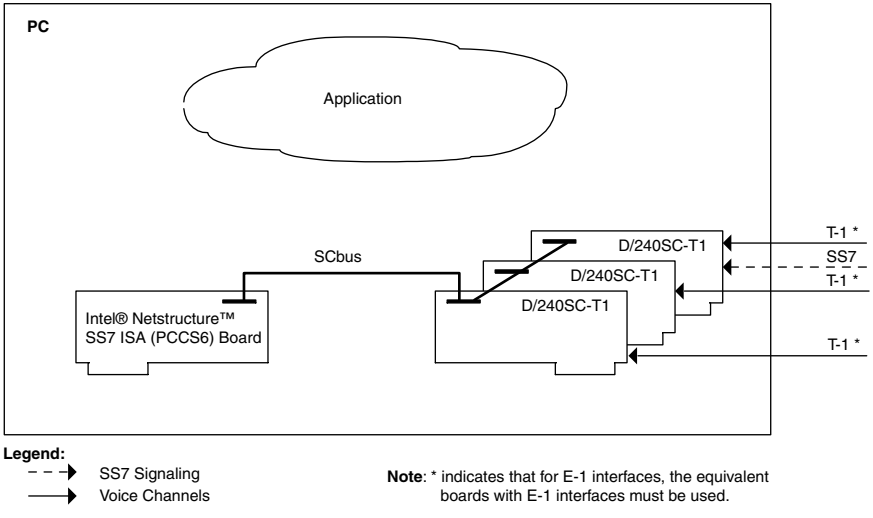


Figure 4. Intel® NetStructure™ SS7 ISA (PCSS6) Board Configuration 2

The key features in this configuration are:

- SS7 link and bearer channels enter through Intel® Dialogic network interface board
- All voice and data resources managed by Intel® Dialogic boards
- T-1/E-1 with SS7 signaling connects to a voice board
- The SS7 signaling is routed to the Intel® NetStructure™ SS7 ISA (PCCS6) board via the SCbus

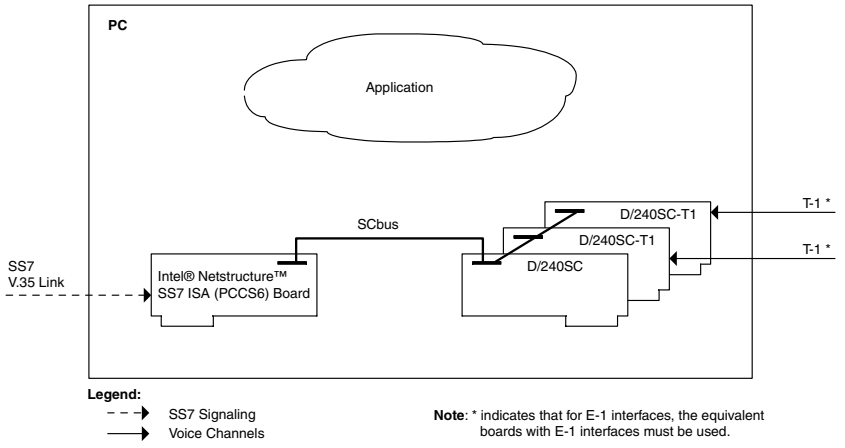


Figure 5. Intel® NetStructure™ SS7 ISA (PCSS6) Board Configuration 3

The key features in this configuration are:

- All T-1/E-1 trunks (bearing voice circuits) enter through Intel® Dialogic network interface boards
- The SS7 link is via a synchronous V.35 connection

NOTE: The V.35 signaling is actually done via two V.11 ports using a using 26-pin D-type connector.

See the documentation accompanying the Intel® NetStructure™ SS7 ISA (PCCS6) board for more detailed information.

NOTE: Global Call SS7 does not support multiple Intel® NetStructure™ SS7 boards in the same system. The SIU is the preferred solution for configuring multi-board configurations.

2.1.2. Signal Interface Unit (SIU)

An Intel-based *black-box* SS7 signaling server. Two models are available, the DSC131 and DSC231. The capacity of each SIU type is shown in Table 2.

Table 2. Capacity of SIUs

	DSC131	DSC231
Signalling cards	2	12
Links	6	32
Linksets	6	8
Call Rate	100/sec.*	100/sec.*

* Call rates can depend on issues in the network such as the way in which signaling is presented. The values should not be considered absolute.

SS7 signaling is extracted from the E-1 or T-1 trunks into the system and the voice circuits can be passed transparently to the outgoing E-1 or T-1 ports. Alternatively, signaling can be connected using V.35 serial links. Signaling information is automatically distributed by the SIU, via TCP/IP, to the host that controls the telephony circuits. Typically this is the system where the voice trunks are terminated on Intel® Dialogic interface boards.

Two SIUs can be configured to share the same point code, providing fully resilient operation within a single point code. In normal operation, signaling can be load-shared across the two SIUs. Then, if one unit fails, the remaining unit handles all signaling. Multiple hosts can be connected to a single SIU, or to a resilient SIU pair, allowing large systems to be built.

Figure 6, Figure 7 and Figure 8 show some configurations using the SIU in conjunction with Intel® Dialogic boards. Table 3 summarizes the features and benefits of each configuration.

Table 3. SIU Configurations - Features and Benefits

Configuration	Features	Benefits
SIU Configuration 1	• V.35 SS7 connection to SIU (DCS131 or DCS231) • Additional T-1/E-1 B channels are connected to voice resources on media servers • SS7 signaling terminated on an SIU • SIU distributes SS7 information to media servers over TCP/IP	• Manage greater number of channels than a single card • Reduced maintenance cost due to smaller overhead relative to management of more circuits
SIU Configuration 2	• SS7 E-1/T-1 connected to SIU (DSC131 or DSC231) • SS7 signaling terminated on SIU • Voice channels routed through SIU via "drop and insert" E-1/T-1 • SIU distributes SS7 information to media servers over TCP/IP	• Additional T-1/E-1 B channels available for voice resources on media servers
SIU Configuration 3	• SS7 link interconnects SIUs to provide a reliable management channel • Dual SS7 links to separate SIUs • SS7 distributed through a single or separate TCP/IP connection	• Provides dual point code management • Redundant SS7 links for back-up of signaling connections

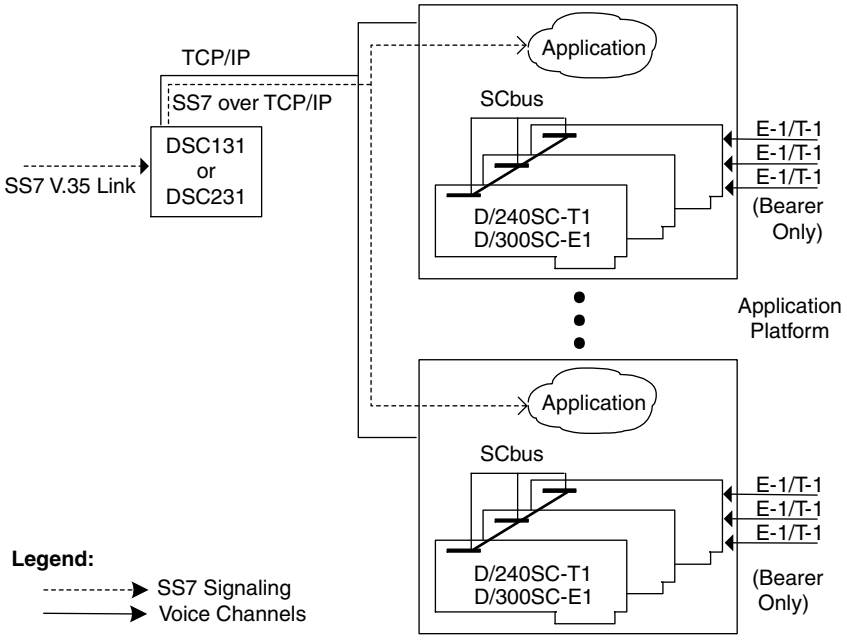


Figure 6. SIU Configuration 1

The key features in this configuration are:

- V.35 SS7 connection to SIU (DCS131 or DCS231)
- T-1/E-1 voice channels are connected to voice resources on media servers
- SS7 signaling terminated on an SIU
- SIU distributes SS7 information to media servers over TCP/IP

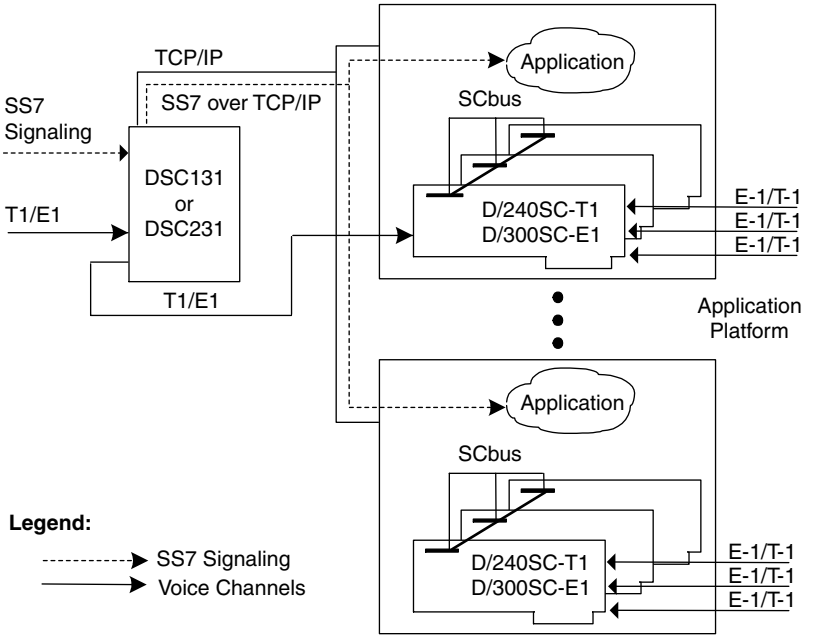


Figure 7. SIU Configuration 2

The key features in this configuration are:

- SS7 connected with E-1/T-1 bearer channels to SIU (DSC131 or DSC231)
- E-1/T-1 voice channels connected to voice resources on media servers
- SS7 signaling terminated on SIU
- B channels routed through SIU via "drop and insert" E-1/T-1
- SIU distributes SS7 information to media servers over TCP/IP

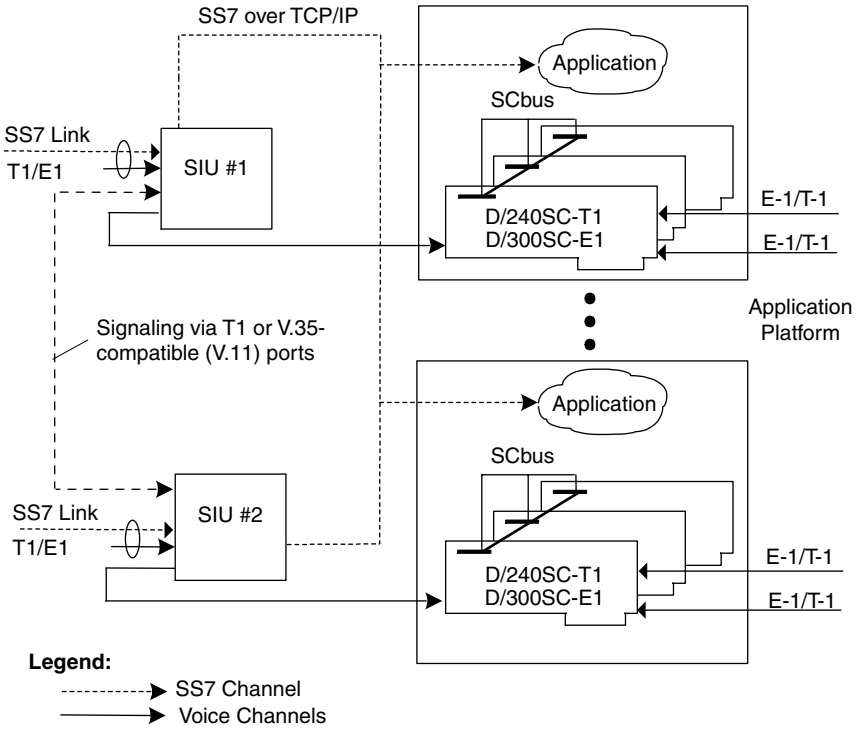


Figure 8. SIU Configuration 3

The key features in this configuration are:

- SS7 link interconnects SIUs to provide a reliability management channel (for single point code management)
- Dual SS7 links to separate SIUs (for dual point-code management)
- SS7 distributed through a single or separate TCP/IP connection

NOTE: To arrange for this set up, you are using two T-1 or E-1 lines out of the SIU boards. This means that you are using one of the available slots of the SIU to pass the voice channels and signaling back out from one SIU to the other. Therefore, depending on the amount of bandwidth being administrated, you might need additional daughter boards.

See the documentation accompanying the DSC131 or DSC231 product for more detailed information.

2.1.3. SS7 Protocol Stack

The protocol stack is the software that implements the various layers of the SS7 protocol. A suite of SS7 protocols is available and includes:

- Message Transfer Part (MTP)
- ISDN User Part (ISUP)
- Telephony User Part (TUP)
- SCCP
- TCAP

MTP is supplied with all SIUs. MTP is avaliable as an option for the Intel® NetStructure™ SS7 boards. Multiple country and switch variants are also available.

NOTE: MTP and ISUP or TUP run on the SIU.

Each of the user parts can run on the host. See the DataKinetics documentation for detailed information. Global Call SS7 currently supports the ISUP and TUP layers. However, non-call-control related user parts could be accessed using the direct DataKinetics API.

2.2. Architecture Overview

Figure 9 is a high level view of the Global Call software architecture and shows how Global Call is used to provide a common call control interface for a variety of network interface technologies including E-1 CAS, T-1 Robbed Bit, analog, ISDN, R4 on DM3, and SS7.

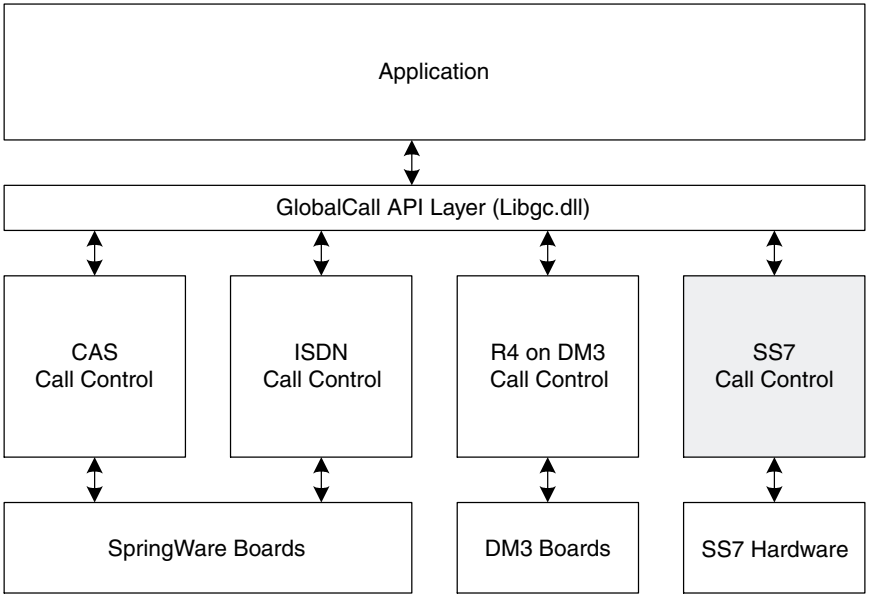


Figure 9. Global Call Architecture

Multiple interface technologies can be mixed within a single application, allowing for example the connection to ISDN and SS7 trunks.

See the *Global Call API Programming Guide* for more information about the overall Global Call architecture.

For SS7, Global Call requires integration with the DataKinetics environment. The environment is based on a number of communicating modules. Each module is a separate task, process, or program (depending on the operating system type) and has a unique identifier called a **module ID**. Modules communicate with each other by sending and receiving messages. Each module has a message queue for the reception of messages. This process is called Inter Process Communication (IPC). See the *DSC131/DSC231 User's Manual* and the *SS7 Programmer's Manual for PCCS6* or the *SS7 Programmer's Manual for SPCI4, SPCI2S and CPM8* for more

information on the software environment and the *System7 Software Environment Programmer's Manual* for more information on IPC.

Global Call SS7 extends this architecture by providing an Intel® Dialogic SS7 server module (with a configurable module ID, typically 0x4d) that can communicate with existing modules. Applications that use Global Call SS7 are also assigned a module ID in the system. This assignment is automatically made by the SS7 server. An example of interaction of the Global Call SS7 software components is shown in Figure 10.

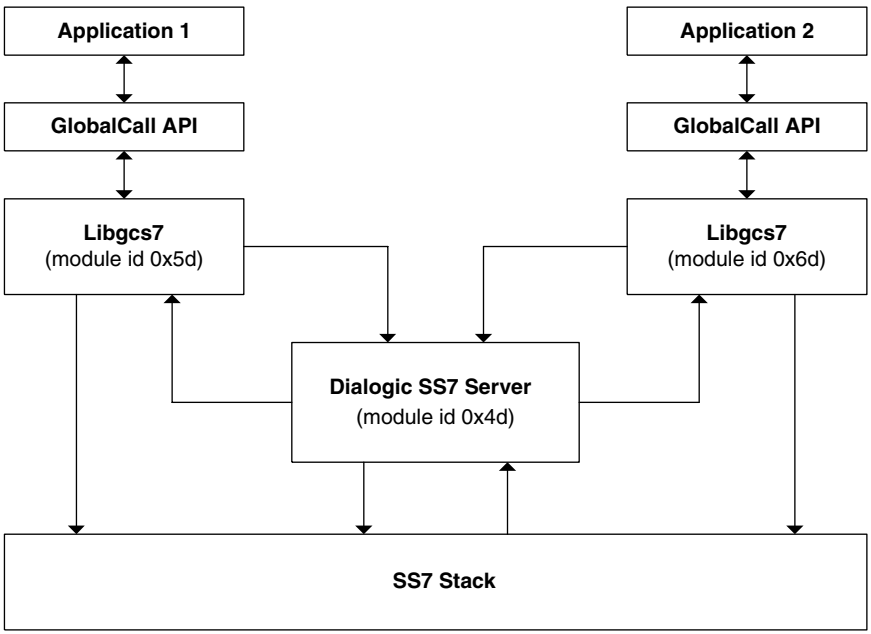


Figure 10. Global Call SS7 Architecture

As shown in the figure, multiple applications can simultaneously use Global Call SS7, provided they do not attempt to control the same line devices (circuits).

The SS7 Call Control Library is called Libgcs7 and is responsible for the communication with other SS7 components in the system. Consequently, an

application using Global Call SS7 does not have to care about any of the lower-level aspects and can be written to the standard Global Call API irrespective of the interface to the SS7 stack, hardware, or communication mechanisms being used. The integration with the actual SS7 stack software environment and the hardware only requires attention during the configuration phase.

For SS7, a Global Call line device maps directly to a telephony circuit in the PSTN. Calls made or received on a circuit are assigned a Call Reference Number (CRN) that is used between the application and the Global Call software to identify the call, just like any other Global Call network interface technology.

2.3. Intel® Dialogic SS7 Server

The Intel® Dialogic SS7 Server is started with all other Intel® Dialogic system components and is responsible for performing the following tasks:

- Assigning module IDs to application processes
- Dispatching messages to application processes
- Automatic handling of dual resilient operations (circuit groups activation and transfer to partner SIU)
- Performing start-up tasks, such as CT Bus transmit time slot assignments for Intel® NetStructure™ SS7 CompactPCI and PCI board systems or SCbus transmit time slot assignments for Intel® NetStructure™ SS7 ISA (PCCS6) board systems)

Module IDs that the SS7 server automatically assigns to applications are taken from a configurable list, allowing maximum flexibility in the use of the DataKinetics software architecture.

The messages dispatched by the SS7 server are handled by Libgcs7, eventually generating standard Global Call events to the application.

In Intel® NetStructure™ SS7 board systems, time slots that are used for voice circuits on trunks connected to the Intel® NetStructure™ SS7 board are automatically assigned a transmit time slot on the CT Bus for Intel® NetStructure™ SS7 CompactPCI or PCI boards, or SCbus for Intel® NetStructure™ SS7 ISA (PCCS6) boards, allowing the application to perform routing of these time slots by using the standard set of bus routing functions,

without having to care about special aspects of interconnecting Intel® NetStructure™ SS7 boards with Intel® Dialogic hardware in the system.

2.4. Global Call SS7 Library

The Global Call SS7 library (Libgcs7) is responsible for performing the following tasks:

- Executing Global Call API functions that are invoked by the application for SS7 line devices
- Handling messages received from the SS7 server
- Sending telephony events, such as call state transitions (for example, GCEV_OFFERED, GCEV_DISCONNECTED etc.), to the application
- Sending ISUP or TUP and management messages to the protocol stack

See Chapter 4, "Applying Global Call Functions to SS7 Applications" for a list of supported Global Call SS7 library functions and how to use them in an SS7 environment.

2.5. SS7 Protocol Stack

The SS7 protocol stack, which consists of the ISUP/TUP layer and the MTP layres, manages the transfer of signal units (some containing messages) between the various layers of the stack and the network.

3. Configuration and Startup

3.1. Configuration Overview

The software configuration of a Global Call SS7 system requires two separate tasks:

- Configuration of the DataKinetics environment
- Configuration of Intel® Dialogic system software to operate with the DataKinetics environment

The first task is to configure the DataKinetics system environment and protocol stack. This is fully described in the DataKinetics documentation, therefore only the basic topics and the topics that are directly important for the Global Call SS7 software are covered in this chapter. Some sample configuration files for various types of configurations are also provided in Appendix C, "Sample Configuration Files".

Configuration of the DataKinetics environment involves the following general activities:

- Configuration of the DataKinetics system environment, which is achieved by editing the *system.txt* file
- Configuration of the SS7 protocol stack, which is achieved by editing the *config.txt* file
- Configuration of the Intel® NetStructure™ SS7 board and optional CT Bus time slot reservation and clocking using DCM.
- Configuration of the Intel® NetStructure™ SIU.
- Alignment of the module IDs in the Intel® Dialogic Configuration Manager (DCM) with the ones defined in *config.txt*.

Figure 11 shows the general activities in sequence.

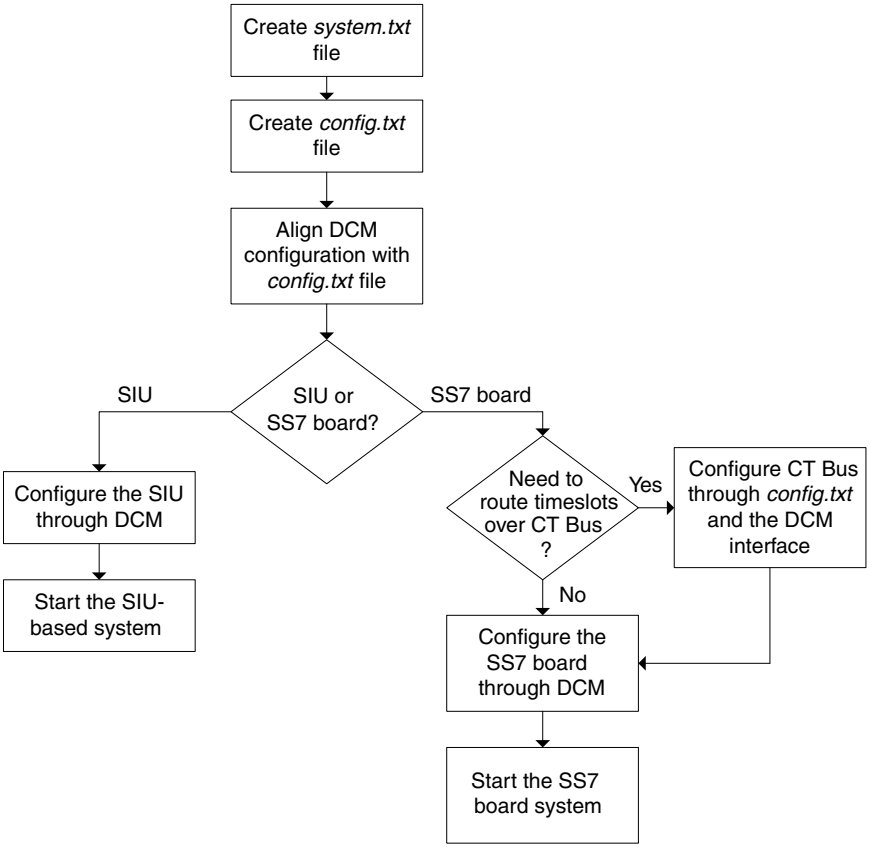


Figure 11. Configuration Work Flow

NOTE: Refer to the DataKinetics documentation for more information on system and protocol configuration sessions.

3.2. System Environment Configuration (system.txt)

The DataKinetics system environment configuration is defined by the *system.txt* file. This file is used by the GCTLOAD program to create message queues and spawn appropriate child processes.

The DataKinetics system uses the concept of modules that send messages to each other. Each module has a unique **module ID**, which must be specified by other modules in order to exchange messages to each other. The module IDs that exist on the host system must be defined using LOCAL commands in the *system.txt* file. Many module IDs are predefined and the lines that specify these modules in the *system.txt* file should be left unchanged.

Some Intel® Dialogic SS7 software components require module IDs that must be defined in the *system.txt* file. For example, the default configuration of the Intel® Dialogic SS7 software requires that the following lines be present in the *system.txt* file:

```
LOCAL 0x4d * Dialogic SS7 Service
LOCAL 0x5d * Dialogic GC/SS7 Application #1
LOCAL 0x6d * Dialogic GC/SS7 Application #2
```

The first line defines the module ID (and corresponding message queue) of the SS7 server. The other two lines define the module IDs reserved for two applications that can use Global Call to control calls on SS7 telephony circuits.

NOTES: 1. The LOCAL command by itself does not specify the intended usage of the defined module ID, it just declares one. For Global Call SS7 to operate properly, there must be at least 2 module IDs declared and left available for its use (one for the SS7 server, and at least one for an application process).

2. In the *system.txt* file, comments are denoted by the asterisk (`*`) symbol, as indicated in the above example.

Other command types that are found in the *system.txt* file are:

- REDIRECT commands. These commands force the DataKinetics runtime system to redirect messages intended for a specific destination module to a different module. For example, in an Intel® NetStructure™ SS7 board system,

this is used to redirect messages for the ISUP module to the module that interfaces with the board (ISUP is running on the board and not on the host).

Besides normal redirections for proper operation of the software (see sample configuration files and the DataKinetics documentation), a system configured for Global Call SS7 should redirect status and management messages to the SS7 server.

In an Intel® NetStructure™ SS7 board system, this is done using the following lines (assuming the SS7 server uses module ID 0x4d, the default value):

```
REDIRECT 0xdf 0x4d    * LIU/MTP2 status messages
```

In an SIU based system, the command is:

```
REDIRECT 0xcf 0x4d    * management messages
```

- FORK_PROCESS commands that tell the GCTLOAD program to spawn child processes. For example, in an Intel® NetStructure™ SS7 ISA (PCCS6) board system, this is used to start the SSD module (or the SSDS module for Intel® NetStructure™ SS7 CompactPCI and PCI boards) that interfaces with the board, and to start the timer modules. On SIU host systems, it can be used to launch the RSI module that is responsible for the TCP/IP communication with the SIU units. A FORK_PROCESS command can also be used to automatically start S7_LOG, a message logging tool that displays system status messages.

  This tool is most useful when proving or debugging a configuration because it provides a visual indication of the PCM trunk status, the link status, and so on. However, it should not be used in parallel with Global Call SS7 because it would intercept some important messages. See Chapter 6, "Troubleshooting" for more information.

  On SIU systems, FORK_PROCESS should only be used to start the RSI module. It should not be used to issue the RSI link activation commands (RSICMD) because these are sent automatically by the SS7 server.

  NOTE: The Intel® Dialogic SS7 software does not require any special FORK_PROCESS commands.

  It is possible to configure the SS7 server to launch the GCTLOAD program automatically. See Section 3.4.2, "Using DCM for General Intel® NetStructure™ SS7 Board Configuration", on page 38 for more information.

In Intel® NetStructure™ SS7 board systems, it is necessary to add the **-i** option to the S7_MGT program in order to complete the startup sequence.

For example:

```
FORK PROCESS S7 MGT -i0x4d    * notify Dialogic SS7 service
```

NOTE: The module ID specified in the example shown should be that of the SS7 server.

3.3. Protocol Stack Configuration (config.txt)

The SS7 protocol stack is usually configured based on the *config.txt* configuration file. For Intel® NetStructure™ SS7 board systems, this file is used by the S7_MGT program, normally spawned by the GCTLOAD program. The S7_MGT program reads the *config.txt* file and sends corresponding configuration messages to the protocol stack modules.

The SIU is configured in two stages:

- Selection of protocol modules and assignment as either SIUA or SIUB is achieved using the **CNSYS** management console command.
- SS7 protocol parameters are set by editing the *config.txt* file.

The following sections describe only aspects of the protocol stack configuration that are important for operation with the Global Call SS7 software. See the DataKinetics documentation for detailed explanations of the all the commands in the *config.txt* file.

3.3.1. TDM Bus Configuration of an Intel® NetStructure™ SS7 Board

For Intel® NetStructure™ SS7 board systems that use the CT Bus (to access voice circuits on a trunk connected to an Intel® NetStructure™ SS7 board, or for routing the SS7 signaling), the CT Bus clocking must be configured.

An Intel® NetStructure™ SS7 board can be configured to take its clock from the CT Bus, acting as an bus “slave”, or to take it from one of its E-1 interfaces and act as a bus “master”, providing the clock for all other boards on the CT bus.

For the Intel® NetStructure™ SS7 ISA (PCCS6) board, the `PCCS6_BOARD` command in the *config.txt* file should have its **flags** argument set as indicated in Table 4.

Table 4. SCbus Clock Configuration for Intel® NetStructure™ SS7 ISA (PCCS6) Boards

Configuration	Flags
SCbus Slave	0x00C2
SCbus Master - clock from LIU1	0x0043
SCbus Master - clock from LIU2	0x0047
SCbus Master - Intel® NetStructure™ SS7 board internal clock	0x0042

For Intel® NetStructure™ SS7 CompactPCI and PCI boards, the `SEPTELCP_BOARD` and `SEPTELPCI_BOARD` commands respectively in the *config.txt* file should have the **flags** argument set to one of the values indicated in Table 5.

Table 5. CT Bus Clock Configuration for Intel® NetStructure™ SS7 CompactPCI and PCI Boards

Configuration	Flags
CT Bus Slave	0x00C2
CT Bus Master - clock from one of the line interfaces	0x0043
CT Bus Master - Intel® NetStructure™ SS7 board internal clock	0x0042

3.3.2. MTP Configuration

The only time where some special MTP configuration is required is when some SS7 link is to be routed over the CT Bus or SCbus, from an Intel® Dialogic network interface card (DTI) to an Intel® NetStructure™ SS7 board. The MTP_LINK lines that correspond to such links must be preceded by a comment having the following syntax:

```
* [link src=<device_name>]
MTP_LINK …
```

where **<device_name>** is the Intel® Dialogic DTI time slot device name (for example, dtiB1T1, dtiB1T31) on which the SS7 signaling link is present.

NOTE: For E-1 lines, physical time slot 16 on a DTI board is usually reserved for signaling, but is named dtiB1T31 (because physical time slot 17 is named dtiB1T16).

Additionally, for links that are routed over the CT Bus, for the Intel® NetStructure™ SS7 ISA (PCCS6) board the **stream** parameter of the MTP_LINK command must always be set to 0x12 and the **timeslot** parameter must be set in such a way that the same time slot of stream 0x12 is not used for more than one signaling link. Normally, you would start with time slot 1 for the first link and increase the number for every additional link routed over the CT Bus. For example, the following lines define two signaling links that must be routed to/from time slots 16 of dtiB1 and dtiB2 boards.

```
                                          stream     ts
* [link src=dtiB1T31]
MTP_LINK 0   0   0   0   0   0    0x12   0x01   0x06
* [link src=dtiB2T31]
MTP_LINK 1   0   1   1   0   1    0x12   0x02   0x06
```

For Intel® NetStructure™ SS7 CompactPCI and PCI boards, the **stream** parameter should be set to 0x83, and the **timeslot** parameter should be set to 0 for the first link, 1 for the second link, 2 for the third link and 3 for the fourth link. The other parameters should be set to the correct values for the link being configured.

```
                                          stream     ts
* [link src=dtiB1T31]
MTP_LINK 0   0   0   0   0   0    0x83     0   0x06
* [link src=dtiB2T31]
MTP_LINK 1   1   0   0   0   1    0x83     1   0x06
```

See Appendix C, “Sample Configuration Files” for more information.

3.3.3. ISUP Configuration

There are two items that require special attention in the ISUP configuration for a system using Global Call SS7 software.

The ISUP_CONFIG command must specify in its UserID argument that the module using the ISUP component is the SS7 server. By default, the SS7 server uses module ID 0x4d.

Additionally, Global Call SS7 relies on a specific type of circuit release procedure in the ISUP module. This is the procedure recommended and it requires that bit 2 (**ISPF_ACR**) and bit 4 (**ISPF_NAI**) of the **<options>** argument of the ISUP_CONFIG command be set to 1. You must also set bit 6 (**ISPF_GSPS**) to 1 for proper generation of **GCEV_BLOCKED** and **GCEV_UNBLOCKED** events.

Consequently, a standard ISUP_CONFIG line for ITU operation looks like the following (assuming Point Code 1 and a maximum of 2 circuit groups):

```
ISUP_CONFIG 1 0x08 0x4d 0x0474 2 64
```

Because an application that uses the Global Call API opens circuits by giving their device name (for example, dtiB1T1 for the first circuit on the first DTI board), Global Call SS7 requires that the *config.txt* file contain some special comments specifying the association between a logical circuit group and the physical device where the circuits are located.

For this purpose, each ISUP_CFG_CGRP command **must be preceded by a comment line** having the following syntax:

```
* [<trunk_mame> <parameters>]
ISUP_CFG_CGRP …
```

where,

- **<trunk_name>** specifies the physical device where the circuits in the group are terminated. This can be a reference to an Intel® Dialogic digital network interface board in which case the name is of the form dtiB*x* (for example, dtiB1, dtiB2, and so on) or one of the trunks on an Intel® NetStructure™ SS7 board in which case the name is dkB1 for the first trunk and dkB2 for the

second trunk. The same name is used as a basis by the application for the network device name when it opens a Global Call SS7 device. See Section 4.2.6, "gc_OpenEx( )", on page 55 for details.

- **<parameters>** can contain zero, one, or more of the following:
  - SIUA or SIUB - only valid for Dual-Resilient SIU configurations, this parameter specifies the default SIU for the group. This is the SIU on which the group should be preferably active (for load-balancing).
  - BaseTs= <baseTs> - specifies the first time slot of the trunk that corresponds to the first circuit of the group. This time slot number is a true physical time slot number (1-31, for E-1). If omitted, the first time slot is assumed.

  NOTE: The BaseTs parameter is especially useful when running ANSI ISUP over E-1 trunks with, for example, two groups of 15 circuits on each E-1 trunk; the second circuit group would be configured with the same <trunk_name> as the first one but with BaseTs=17.

The following fragment shows an example of circuit groups configuration where the first group is located on the dtiB1 board and has SIU A as preferred server, while the second group is on dtiB2 and has SIU B as preferred server:

```
* [dtiB1 SIUA]
ISUP_CFG_CCTGRP  0  1  0x01  0x01  0x7fff7fff  0x0002 0x00
* [dtiB2 SIUB]
ISUP_CFG_CCTGRP  1  1  0x21  0x21  0x7fff7fff  0x0002 0x00
```

For an example of the **<options>** parameter in the ISUP_CFG_CGRP command, see the sample configuration files in Appendix C, "Sample Configuration Files". Also, see the *System7 ISUP Programmer's Manual* for more information.

3.3.4. TUP Configuration

TUP configuration is achieved in much the same way as the ISUP configuration described in Section 3.3.3, "ISUP Configuration", on page 32 with the following differences:

- In the *system.txt* file, there should be a REDIRECT command for the TUP module as follow:

```
REDIRECT    0x4A    0x20   *TUP Module
```

- In the *config.txt* file, the appropriate binary should be downloaded and the corresponding license applied. The following are some examples:

  For an Intel® NetStructure™ SS7 ISA (PCCS6) board system:

  ```
  PCCS6_BOARD 0 0 0 0X0042 tup76.dc2
  ```

  For an Intel® NetStructure™ SS7 CompactPCI or PCI board system:

  ```
  SEPTELPCI_BOARD 0 0X0042 SS7.dc3 TUP-L
  ```

  See the DataKinetics documenation for more information.

- TUP parameters must be configured. The TUP_CONFIG command is described in the *SS7 Programmer's Manual for SPCI4, SPCI2S and CPM8* and the *SS7 Programmer's Manual for PCCS6*.

  For example:

  ```
  * TUP_CONFIG <reserved><reserved><user_id><options>
    <num_grps> <num_ccts>

  TUP_CONFIG 0 0 0x4d 0x8166 128 4096
  ```

  The **options** parameter is a 16-bit value containing global run-time options for the operation of the TUP module. The meaning of each bit is as defined for the **options** parameter in the TUP *Configuration Request* message described in the *System7 TUP Programmer's Manual*. For Global Call SS7 to function correctly, the following bits in the options argument **must** be set:

  - bit 5 (TUPF_GSPS)
  - bit 6 (TUPF_ACR)
  - bit 15 (TUPF_NAI)

- Global Call SS7 requires that the *config.txt* file contain some special comments specifying the association between a logical circuit group and the physical device on which the circuits are located. An explanation of the usage of these comments for ISUP_CFG_CGRP is described in Section 3.3.3, "ISUP Configuration", on page 32. The information in that section applies when using the TUP_CFG_CCTGRP command for TUP configuration also. For example:

```
* Define TUP circuit groups:
* TUP_CFG_CCTGRP <gid><dpc><base_cic><base_cid>
  <cic_mask> <options><user_inst><user_id><opc><ssf>
*[dtiB1]
TUP_CFG_CCTGRP 0 2 0x01 0x01 0x7fff7fff 0x0070 0
  0x4d 1 0x08
*[dtiB2]
TUP_CFG_CCTGRP 1 2 0x21 0x21 0x7fff7fff 0x0070 0
  0x4d 1 0x08
```

3.4. Using DCM to Configure Intel® NetStructure™ SS7 Boards

For Intel® NetStructure™ SS7 board configuration, you must consider whether you want to use the CT Bus for time slot routing on the SS7 board:

- If you need to have circuits from a trunk of the Intel® NetStructure™ SS7 board available on the CT Bus, or if an SS7 signaling link has been routed over the CT Bus between a Intel® Dialogic network interface board and the Intel® NetStructure™ SS7 board, you must configure the SS7 board as described earlier in Section 3.3.1, “TDM Bus Configuration of an Intel® NetStructure™ SS7 Board”, on page 29 and refer to Section 3.4.1, “Using the Third Party Tech Option in DCM to Manage the CT Bus”, on page 35 in order to reserve the correct range of timeslots on the CT Bus.
- If you do not need to use the CT Bus for time slot routing on the SS7 board, you can skip the procedure referenced above and refer directly to Section 3.4.2, “Using DCM for General Intel® NetStructure™ SS7 Board Configuration”, on page 38.

3.4.1. Using the Third Party Tech Option in DCM to Manage the CT Bus

If the Intel® NetStructure™ SS7 board is required to share TDM bus resources with Intel® Dialogic boards, the system software needs to identify the Intel® NetStructure™ SS7 board’s TDM bus status (primary master, slave) and the TDM bus time slots the device will be using. This prevents Intel® Dialogic boards from using time slots that are to be specifically reserved for the Intel® NetStructure™ SS7 board. DCM’s Add Hardward Wizard enables this configuration.

Use the following procedure to reserve TDM bus resources for an Intel® NetStructure™ SS7 board:

1. In the DCM Main Window, click **Add Device** from the **Action** pull-down menu. This activates the Add Hardware Wizard.
2. At the Choose Family and Model screen, select **ThirdPartyTech** from the list of device families. **ThirdPartyDevice** is automatically selected in the list of models. Click **Next**.
3. At the Name the Board screen, enter a unique identifier for the Intel® NetStructure™ SS7 board. This identifier will appear in the DCM Main Window.
4. Click the **Next** button to display the **Third Party TDM Bus Capabilities** screen. This screen allows you to define the TDM bus capabilities of the Intel® NetStructure™ SS7 board and indicate if the board will serve as the current Primary Master for the system.
5. In the **TDM Bus Master/Slave Capabilities** section, proceed as follows:
   a. If the Intel® NetStructure™ SS7 board will *always* operate as a slave in an SCbus system, select the **SC Slave capable** checkbox and proceed with step 7.
   b. To make the Intel® NetStructure™ SS7 board *capable* of being the Primary Master in an SCbus system, select the **SC Master capable** checkbox. If you would like to *define* the Intel® NetStructure™ SS7 board as the Primary Master for the system, proceed with step 6. Otherwise, continue with step 7.

   NOTE: Any board that is capable of being a Primary Master must also be capable of being a slave. Therefore, the system automatically selects the **SC Slave capable** checkbox whenever the **SC Master capable** checkbox is selected.

   c. If the Intel® NetStructure™ SS7 board will *always* operate as a slave in an H.100 (PCI) system, select the **H.100 Slave capable** checkbox and proceed with step 7.
   d. To make the Intel® NetStructure™ SS7 board *capable* of being the Primary Master in an SCbus system, select the **H.100 Master capable** checkbox. If you would like to *define* the Intel® NetStructure™ SS7 board as the Primary Master for the system, proceed with step 6. Otherwise, continue with step 7.

NOTE: Any board that is capable of being a Primary Master must also be capable of being a slave. Therefore, the system automatically selects the **H.100 Slave capable** checkbox whenever the **H.100 Master capable** checkbox is selected.

e. If the Intel® NetStructure™ SS7 board will *always* operate as a slave in an H.110 (CompactPCI) system, select the **H.110 Slave capable** checkbox and proceed with step 7.

f. To make the Intel® NetStructure™ SS7 board *capable* of being the Primary Master in an SCbus system, select the **H.110 Master capable** checkbox. If you would like to *define* the Intel® NetStructure™ SS7 board as the Primary Master for the system, proceed with step 6. Otherwise, continue with step 7.

NOTE: Any board that is capable of being a Primary Master must also be capable of being a slave. Therefore, the system automatically selects the **H.110 Slave capable** checkbox whenever the **H.110 Master capable** checkbox is selected.

6. To assign the Intel® NetStructure™ SS7 board as the system's current Primary Master, select the checkbox in the **TDM Bus Primary Master Assignment** section.

   NOTE: If you assign Intel® NetStructure™ SS7 board as the current Primary Master for the system, you must start the Intel® NetStructure™ SS7 board before using the DCM to start the Intel Dialogic system service.

7. Click the **Next** button to display the **Third Party Time Slot Allocation** screen.

8. Use the **Starting** and **Ending** boxes to enter a range of time slots for use by the Intel® NetStructure™ SS7 board. For example, if you would like to reserve time slots 1 to 124 for the Intel® NetStructure™ SS7 board, enter 1 in the **Starting** box and 124 in the **Ending** box.

   NOTE: For an Intel® NetStructure™ SS7 board, you must select only *one* range of CT Bus time slots and that range must be the *first* range in the list of allocated ranges.

9. Click the **Reserve** button. The system reserves time slots with the ranges specified in step 8 for exclusive use by the Intel® NetStructure™ SS7 board. Intel Dialogic devices cannot use the reserved time slots. If the system cannot reserve the time slot range you selected, an error message will be displayed giving the option to view the system event log.

NOTE: An allocated time slot range can be deallocated by checking the check box to the left of the range and clicking the **Release** button.

10. Click the **Finish** button to end the procedure.

3.4.2. Using DCM for General Intel® NetStructure™ SS7 Board Configuration

Configure the Intel® NetStructure™ SS7 board using the Intel® Dialogic Configuration Manager (DCM) as follows:

1. From the **Action** menu in the main DCM dialog box, choose **Add Device**. The DCM - Add Hardware Wizard is displayed.
2. Select **SS7** as the device from the **Family** scroll box.
3. Select **Septel card** in the **Model** scroll box.
4. Click **Next**.
   The DCM - Add Hardware Wizard prompts you for a name.
5. Enter a name for the Intel® NetStructure™ SS7 board. This is not used by the software but must be provided for proper operation of DCM. Then, click **Next**.
6. Set the parameters.
   - From the **System** tab, you **must** set the following parameters:
     - In the **Value** field for the **ConfigDir** parameter, enter the path to your *config.txt* file. This must be the same directory that the DataKinetics software uses to obtain its configuration. Leave all other fields at their default values.
     - **ServiceModuleID** defines the module ID used by the SS7 server. This must be one of the module IDs declared LOCAL in the *system.txt* file. Default value: 0x4d. See Section 3.2, "System Environment Configuration (system.txt)", on page 27.
   - Optionally, set the following parameters:
     - **Auto Links Activation** controls the activation of SS7 MTP links. If set to its default value, None, the links must be activated using the MTPSL command (see Section 3.5, "Starting an Intel® NetStructure™ SS7 Board System", on page 40). Set to All to request the SS7 server to activate all the links during its startup
     - **LogLevels** controls the generation of SS7 server logging information. If set to All, the SS7 server will produce a log file that can be very

useful in troubleshooting a system. See Chapter 6, “Troubleshooting” for more on this topic. The default value is All.

- **LogMaxLines** restricts the maximum length of an SS7 server’s log file. The default value is 20000.
- **AppModuleIDs** defines the module IDs that the SS7 server can assign to applications. These must be declared LOCAL in the *system.txt* file. The format of the parameter is a space delimited list of module IDs. The default values are listed as 0x5d 0x6d.
- **GCTLOAD Control** determines if the GCTLOAD program should run automatically at startup. If set to Yes, the SS7 server will try to start the GCTLOAD program automatically. In this case, it is necessary to add the `-i` option to the S7_MGT program in order to complete the startup sequence (see Section 3.2, “System Environment Configuration (system.txt)”, on page 27) for more information). The default value is No.

NOTE: This option should only be activated **after** you have adapted and **fully tested** your configuration since the GCTLOAD window, which provides very useful configuration debugging information, is no longer displayed when this option is selected.

- **GCTLOAD Path** contains the path to the GCTLOAD program file. This field must be set if the **GCTLOAD Control** parameter is set to Yes.

NOTE: This field can be left empty if the GCTLOAD program resides in the same directory as the *config.txt* file (ConfigDir option).

- **WatchDogMaxTime** defines the maximum time that the SS7 server waits for keep-alive notifications from the call control library. This mechanism allows the SS7 server to detect when an application has crashed or has exited without proper shutdown, and then frees for reuse the module ID to which the application was assigned. The default value is 8 seconds.

NOTE: During step-by-step debugging sessions, the maximum watchdog time may easily be exceeded, causing the SS7 server to stop routing messages to the application process. Therefore, during debugging sessions, you should set the **WatchDogMaxTime** parameter to a higher value than the default.

When you have set all options to the desired values click the **OK** button.

7. If the Intel® NetStructure™ SS7 board is connected to the CT Bus and you have performed the procedures in Section 3.4.1, "Using the Third Party Tech Option in DCM to Manage the CT Bus", on page 35, the name you entered as a third party device identifier should appear in the DCM tree view under the **ThirdPartyTech** group.

   The Intel® NetStructure™ board can be set to act as a CT Bus master as follows:

   a. Under the **TDM Bus** group, double-click **Bus-0** to open the **Properties for Bus-0** dialog.
   b. On the **TDM Bus Configuration** tab, click on the **Primary Master FRU (User Defined)** parameter to populate the **Parameter** and **Value** fields.
   c. In the **Value** field, select the name of the third party device identifier and click the Apply button. The value you enter in this field must correspond with the setting in the *config.txt* file that are described in Section 3.3.1, "TDM Bus Configuration of an Intel® NetStructure™ SS7 Board", on page 29.
   d. Click **OK** to close the dialog.
   e. Exit DCM.

3.5. Starting an Intel® NetStructure™ SS7 Board System

The Intel® Dialogic system service downloads the required firmware to Intel® Dialogic boards, starts all Intel® Dialogic device drivers, and assigns CT Bus time slots.

Starting the system involves two steps:

1. Start the DataKinetics environment. This involves starting the GCTLOAD program, which sets up the IPC (Inter Process Communication) and messaging system.

   The GCTLOAD program also launches S7_MGT that reads the *config.txt*, downloads the specified firmware to the board and configures the stack as specified in the *config.txt* file.

NOTE: The SS7 links must be activated using the MTPSL command. For example, to activate the first link of the linkset with ID 0, issue the following command: MTPSL ACT 0 0. If the **Auto Links Activation** parameter (see Section Section 3.4.2, "Using DCM for General Intel® NetStructure™ SS7 Board Configuration", on page 38) is set to All, the SS7 server will perform the activation of all links during its startup.

2. Start the Intel® Dialogic system service. The Intel® Dialogic system service can be started from the Intel® Dialogic Configuration Manager (DCM) and it automatically performs all initialization steps required by the Global Call SS7 system (excluding the preceding step 1).

3.6. Global Call SS7 Configuration of SIUs

The Global Call SS7 software is configured using the Intel® Dialogic Configuration Manager (DCM) as follows:

1. From the **Action** menu in the main DCM dialog box, choose **Add Device**. The DCM - Add Hardware Wizard is displayed.
2. Select **SS7** as the device from the **Family** scroll box.
3. Select the **Septel SIU** in the model box.
4. Click **Next**.
   The DCM - Add Hardware Wizard prompts you for a name.
5. Enter a name for the SIU server. This is not used by the software but must be provided for proper operation of DCM.
6. Click **Next**.
7. Set the parameters:
   - From the **System** tab, you **must** set the following parameters:
     - Enter the **HostID** of the system. If there is only one host connected to the SIU(s), select ID 0.
     - **ServiceModuleID** defines the module ID used by the SS7 server. This must be one of the module IDs declared LOCAL in the *system.txt* file. Default value: 0x4d. See Section 3.2, "System Environment Configuration (system.txt)", on page 27.
     - Other fields in the **System** tab may be left at their default values.
   - From the **SIU Server** tab, you **must** set the following parameters.

- Enter the IP address of the SIU.
- Specify the login and password for a valid FTP account on the SIU. SIUs provided more recently come with an FTP user account predefined (login "ftp" and password empty). For the SIU520, the login and password are both set to "siuftp" by default. See "Configuration procedure" in the *Intel® NetStructure™ SIU520 Developer's Manual* for more information. If required, also specify, in the **RemoteConfigDir** field, the directory on the SIU on which the *config.txt* file is located (for example, **/home/dklsiu**). For SIUs provided more recently, when using the predefined FTP account, you can leave this field at its default value, which is the dot (.) character.

NOTE: If you are configuring a Dual-Resilient SIU system, you must add an SS7 device for each SIU that will have its own IP address (and potentially its own FTP account and directory data). The first configured SIU must be SIU A, and the second SIU B (see the DataKinetics documentation for more information about dual-resilient SIU configurations).

Setting the following parameters is optional:

- **LogLevels** controls the generation of SS7 server logging information. If set to All, the SS7 server will produce a log file that can be very useful in troubleshooting a system. See Chapter 6, "Troubleshooting" for more on this topic. The default value is All.
- **LogMaxLines** restricts the maximum length of an SS7 server's log file. The default value is 20000.
- **AppModuleIDs** defines the module IDs that the SS7 server can assign to applications. These must be declared LOCAL in the *system.txt* file. The format of the parameter is a space delimited list of module IDs. The default values are listed as 0x5d snd 0x6d.
- **GCTLOAD Control** determines if the GCTLOAD program should run automatically at startup. If set to Yes, the SS7 server will try to start the GCTLOAD program automatically. The default value is No.

NOTE: This option should only be activated **after** you have adapted and **fully tested** your configuration since the GCTLOAD window, which provides very useful configuration debugging information, is no longer displayed when this option is selected.

- **GCTLOAD Path** contains the path to the GCTLOAD program file. This field must be set if the **GCTLOAD Control** parameter is set to Yes.
- **WatchDogMaxTime** defines the maximum time that the SS7 server waits for keep-alive notifications from the call control library. This mechanism allows the SS7 server to detect when an application has crashed or has exited without proper shutdown, and then frees for reuse the module ID to which the application was assigned. The default value is 8 seconds.

  NOTE: During step-by-step debugging sessions, the maximum watchdog time may easily be exceeded, causing the SS7 server to stop routing messages to the application process. Therefore, during debugging sessions, you should set the WatchDogMaxTime parameter to a higher value than the default.

From the **Dual Resilient** tab:

- **SiuCommandTimeout** specifies the timeout value to use when waiting for group (de)activation command responses from an SIU. The default value is 5 seconds.
- **SiuUpDebounceTime** specifies the time to use when detecting SIU availability. This debounce avoids undertaking unnecessary actions in case of intermittent TCP/IP connection failures. The default value is 8 seconds.
- **MaxCmdRetries** specifies the maximum number of times the SS7 server reattempts sending a group (de)activation command to an SIU before declaring failure. A resend is required when the SIU is already performing a command for another host system. The default value is 5 attempts.

From the **Misc** tab:

- **FtpRetries** defines the number of times the SS7 server will reattempt to get the *config.txt* file from an SIU.
- **FtpTimeout** defines the maximum time to wait for a response from an SIU while getting the *config.txt* file via FTP.
- **SiuInitTimeout** defines the maximum time that the SS7 server will wait at startup for an SIU to come on-line before considering it as being down.

3.7. Starting an SIU-based System

When you start an SIU-based system, the Intel® Dialogic system service downloads the required firmware to Intel® Dialogic boards, starts all Intel® Dialogic device drivers, and assigns time slots.

CAUTION

At least one SIU must be up and running when you start the service. This is required because the configuration is read from the SIU.

Starting the system involves two steps:

1. Start the DataKinetics environment. This involves starting the `gctload` program, which sets up the IPC (Inter Process Communication) and messaging system.

   If the *system.txt* file is correctly configured, the `GCTLOAD` program loads the RSI module responsible for communicating with the server(s). However, the actual connection to the server(s) is made by the Intel® Dialogic SS7 server.
2. Start the Intel® Dialogic SS7 server. The Intel® Dialogic system service can be started from the Dialogic Configuration Manager (DCM) and it automatically performs all initialization steps required by the Global Call SS7 system (excluding the preceding step 1).

4. Applying Global Call Functions to SS7 Applications

4.1. Global Call Function Support

Table 6 lists the Global Call functions that are either supported, not supported, or supported with enhancements or differences from the standard support. See the *Global Call API Library Reference* for more information on each function.

Table 6. Function Support

Function	Level of Support
gc_AcceptCall( )	Supported with variances. See Section 4.3.1, “gc_AcceptCall( )”, on page 59.
gc_AlarmName( )	Not supported.
gc_AlarmNumber( )	Not supported.
gc_AlarmNumberToName( )	Not supported.
gc_AlarmSourceObjectID( )	Not supported.
gc_AlarmSourceObjectIDToName( )	Not supported.
gc_AlarmSourceObjectName( )	Not supported.
gc_AlarmSourceObjectNameToID( )	Not supported.
gc_AnswerCall( )	Supported with variances. See Section 4.3.2, “gc_AnswerCall( )”, on page 59.
gc_Attach( )	Supported.
gc_AttachResource( )	Not supported. Use **gc_Attach( )**.
gc_BlindTransfer( )	Not supported.

Table 6. Function Support (Continued)

Function	Level of Support
gc_CallAck( )	Supported with variances. See Section 4.3.3, "gc_CallAck( )", on page 59.
gc_CallProgress( )	Not supported.
gc_CCLibIDToName( )	Supported.
gc_CCLibNameToID( )	Supported.
gc_CCLibStatus( )	Supported.
gc_CCLibStatusAll( )	Supported.
gc_CCLibStatusEx( )	Supported.
gc_Close( )	Supported.
gc_CompleteTransfer( )	Not supported.
gc_CRN2LineDev( )	Supported.
gc_Detach( )	Supported.
gc_DropCall( )	Supported with variances. See Section 4.3.4, "gc_DropCall( )", on page 60.
gc_ErrorInfo( )	Supported.
gc_ErrorValue( )	Supported. See Section 4.2.1, "gc_ErrorValue( )", on page 52.
gc_Extension( )	Supported. See Section 4.2.2, "gc_Extension( )", on page 52.
gc_GetAlarmConfiguraton( )	Not supported.
gc_GetAlarmFlow( )	Not supported.
gc_GetAlarmParm( )	Not supported.
gc_GetAlarmSourceObjectList( )	Not supported.

Table 6. Function Support (Continued)

Function	Level of Support
gc_GetAlarmSourceObject NetworkID( )	Not supported.
gc_GetANI( )	Deprecated function. Use **gc_GetCallInfo( )**.
gc_GetBilling( )	Not supported.
gc_GetCallInfo( )	Supported with variances. See Section 4.3.5, "gc_GetCallInfo( )", on page 60.
gc_GetCallProgress Parm( )	Not supported.
gc_GetCallState( )	Supported.
gc_GetConfigData( )	Not supported.
gc_CRN( )	Supported.
gc_GetCTInfo( )	Not supported.
gc_GetDNIS( )	Deprecated function. Use **gc_GetCallInfo( )**.
gc_GetFrame( )	Not supported.
gc_GetInfoElem( )	Not supported.
gc_GetLineDev( )	Supported.
gc_GetLineDevState( )	Supported.
gc_GetMetaEvent( )	Supported.
gc_GetMetaEventEx( )	Supported.
gc_GetNetCRV( )	Not supported.
gc_GetNetworkH( )	Supported with variances. See Section 4.2.3, "gc_GetNetworkH( )", on page 53.

Table 6. Function Support (Continued)

Function	Level of Support
gc_GetParm( )	Supported with variances. See Section 4.2.10, "gc_SetParm( )", on page 57.
gc_GetResourceH( )	Not supported.
gc_GetSigInfo( )	Supported. See Section 4.2.5, "gc_GetSigInfo( )", on page 53.
gc_GetUserInfo( )	Not supported.
gc_GetUserAttr( )	Supported.
gc_GetVer( )	Supported.
gc_GetVoiceH( )	Supported.
gc_GetXmitSlot( )	Supported.
gc_HoldAck( )	Not supported.
gc_HoldCall( )	Supported with variances. See Section 4.3.7, "gc_HoldCall( )", on page 61 for more information.
gc_HoldRej( )	Not supported.
gc_LinedevToCCLIBID( )	Supported.
gc_Listen( )	Supported.
gc_LoadDxParm( )	Not supported.
gc_MakeCall( )	Supported with variances. See Section 4.3.8, "gc_MakeCall( )", on page 61.
gc_Open( )	Deprecated function. Use **gc_OpenEx( )**.
gc_OpenEx( )	Supported with variances. See Section 4.2.6, "gc_OpenEx( )", on page 55.

Table 6. Function Support (Continued)

Function	Level of Support
gc_QueryConfigData( )	Not supported.
gc_ReleaseCall( )	Supported.
gc_ReleaseCallEx( )	Supported.
gc_ReqANI( )	Not supported.
gc_ReqMoreInfo( )	Supported.
gc_ReqService( )	Not supported.
gc_ResetLineDev( )	Supported with variances. See Section 4.2.7, “gc_ResetLineDev( )”, on page 56.
gc_RespService( )	Not supported.
gc_ResultInfo( )	Supported.
gc_ResultMsg( )	Supported.
gc_ResultValue( )	Supported with variances. See Section 4.2.8, “gc_ResultValue( )”, on page 56.
gc_RetrieveAck( )	Not supported.
gc_RetrieveCall( )	Supported with variances. See Section 4.3.9, “gc_RetrieveCall( )”, on page 62 for more information.
gc_RetrieveRej( )	Not supported.
gc_SndMoreInfo( )	Supported.
gc_SetAlarmConfiguration( )	Not supported.
gc_SetAlarmFlow( )	Not supported.
gc_SetAlarmNotifyAll( )	Not supported.
gc_SetAlarmParm( )	Not supported.

Table 6. Function Support (Continued)

Function	Level of Support
gc_SetBilling( )	Supported with variances. See Section 4.3.10, "gc_SetBilling( )", on page 62.
gc_SetCallingNum( )	Supported.
gc_SetCallProgressParm( )	Not supported.
gc_SetChanState( )	Supported with variances. See Section 4.2.9, "gc_SetChanState( )", on page 57.
gc_SetConfigData( )	Supported with variances. See Section 4.2.12, "gc_SetConfigData( )", on page 58 for more information.
gc_SetEvtMsk( )	Supported.
gc_SetInfoElem( )	Supported. See Section 4.3.11, "gc_SetInfoElem( )", on page 63.
gc_SetParm( )	Supported with variances. See Section 4.2.10, "gc_SetParm( )", on page 57.
gc_SetupTransfer( )	Not supported.
gc_SetUserInfo( )	Not supported.
gc_SetUserAttr( )	Supported.
gc_SndFrame( )	Not supported.
gc_SndMsg( )	Supported. See Section 4.2.13, "gc_SndMsg( )", on page 58.
gc_Start( )	Supported.
gc_StartTrace( )	Supported with variances. See Section 4.2.11, "gc_StartTrace( )", on page 57.

Table 6. Function Support (Continued)

Function	Level of Support
gc_Stop( )	Supported.
gc_StopTrace( )	Supported with variances. See Section 4.2.14, "gc_StopTrace( )", on page 59.
gc_StopTransmitAlarms( )	Not supported.
gc_SwapHold( )	Not supported.
gc_TransmitAlarms( )	Not supported.
gc_UnListen( )	Supported.
gc_util_delete_parm_blk( )	Supported.
gc_util_find_parm( )	Supported.
gc_util_insert_parm_ref( )	Supported.
gc_util_insert_parm_val( )	Supported.
gc_util_next_parm( )	Supported.
gc_WaitCall( )	Supported.

For SS7, all the Global Call API functions that have a mode argument must be used in asynchronous mode, except the routing functions (**gc_Listen( )**, **gc_UnListen( )**, and **gc_GetXmitSlot( )**) that must be used in synchronous mode.

The SS7 specific constants and data structures are defined in the *Libgcs7.h* and *cc_s7.h* header files. An application should only include *Libgcs7.h* (*cc_s7.h* being included by the latter).

4.2. System and Line Device Related Functions

The SS7 call control library supports the line-related Global Call functions with the additional functionality or specific aspects described in the following subsections.

4.2.1. gc_ErrorValue( )

The SS7 call control library provides both standard Global Call error codes and SS7-specific error codes (*cclib_errorp* argument), which are useful when diagnosing function failures. See Appendix A, "SS7-Specific Error Codes" for more information. The error codes are also listed in the *cc_s7.h* header file.

NOTE: The **gc_ErrorValue( )** function is deprecated. The preferred alternative is **gc_ErrorInfo( )**.

4.2.2. gc_Extension( )

The **gc_Extension( )** function and corresponding GCEV_EXTENSION event is primarily used to support the Continuity Check feature.

For the GCEV_EXTENSION event, the extevtdatap field of the METAEVENT structure is a pointer to an EXTENSIONEVTBLK structure. The ext_id member of EXTENSIONEVTBLK can be:

- S7_EXT_CONTCHECK - Indicating the beginning of a Continuity Check.
- S7_EXT_CONTCHECK_END - Indicating the end of a Continuity Check

The parmblk field of the EXTENSIONEVTBLK structure contains additional information. The parmblk field is of type GC_PARM_BLK and contains only a GC_PARM_DATA structure. The set_ID of GC_PARM_DATA is S7SET_CONTCHECK, and the parm_ID is S7PARM_CONTCHECK_TYPE. The parm_data_size is sizeof(int).

For an inbound Continuity Check request, the value of this parameter is S7RV_CC_INBOUND. See Section 5.10.1, "Inbound Continuity Check", on page 73 for more information.

For an outbound, out-of-call Continuity Check request, the application can use the **gc_Extension( )** function with an **ext_id** of S7_EXT_REQUESTCONTCHECK. See Section 5.10.2, "Outbound Continuity Check", on page 75 for more information.

For an outbound, in-call Continuity Check request, the application **must** use the **gc_MakeCall( )** function. See Section 4.3.8, “gc_MakeCall( )”, on page 61 for more information.

4.2.3. gc_GetNetworkH( )

The **gc_GetNetworkH( )** function can be used to retrieve the network device handle associated with the line device. For circuits located on an Intel® Dialogic network interface board (DTI), the returned handle can then be used when invoking Dialogic DTI functions.For circuits located on an Intel® NetStructure™ SS7 board, the returned handle will be the same as the specified line device. This handle cannot be used with DTI functions.

Typical usage of this function was to perform routing of a Global Call line device (**dt_listen( )**, **dt_getxmitslot( )**). However, this call control library supports the Global Call functions (**gc_Listen( )**, **gc_GetXmitSlot( )**) that can be used regardless of the type of network interface device (DTI or SS7). Therefore, for routing of SS7 line devices, it is recommended to always use the Global Call functions instead of the DTI functions. This makes the network device type transparent to the application.

See Section 5.4, “Routing”, on page 67 later in this document for more on routing.

4.2.4. gc_GetParm( )

The **gc_GetParm( )** function can be used to retrieve the following parameters:

- **GCPR_CALLINGPARTY** - Default Calling Party Address.
- **GCPR_MINDIGITS** - The minimum number of digits to collect before reporting an OFFERED call.
- **GCPR_RECEIVE_INFO_BUFFER** - The size of the cyclic buffers used to store messages that can be retrieved using the **gc_GetSigInfo( )** function. See Section 4.2.5, “gc_GetSigInfo( )”, on page 53 for details.

4.2.5. gc_GetSigInfo( )

The **gc_GetSigInfo( )** function enables an application to retrieve the content of the message that triggered an event. This can be used if the application requires access

to some SS7 specific message parameter that is not directly accessible using another Global Call function. It is then up to the application to parse the message and extract the information it requires.

Since events are delivered to the application using an asynchronous mechanism (SRL event queue), it is possible that a subsequent message may already be received and other events already be put in the queue by the time the application calls the **gc_GetSigInfo( )** function. Therefore the SS7 call control library stores messages in a cyclic buffer so that the application can retrieve a message associated with a particular event. The event for which the application wishes to retrieve the associated message is specified by passing the Global Call metaevent to the function.

The size of the cyclic buffer is configurable by using the **gc_SetParm( )** function and specifying the **GCPR_RECEIVE_INFO_BUF** parameter. There is one cyclic buffer for each circuit. The default size of the buffer being 0, an application that wishes to use the **gc_GetSigInfo( )** function **must** set the GCPR_RECEIVE_INFO_BUF parameter for each line device.

NOTE: The third parameter in the **gc_GetSigInfo( )** function signature, **info_id**, is ignored by the SS7 call control library.

The retrieved messages are returned as an **S7_SIGINFO_BLK** structure in the buffer provided by the application. See Section 7.4, "S7_SIGINFO_BLK Structure", on page 98. The structure contains the ISUP primitive (for example, IAM, ACM, ANM…) and the message parameters. The message parameters are in a format that corresponds very closely to the ISUP format, the difference being that all parameters are formatted the same way whether they are mandatory or optional in the message. All parameters follow the format of ISUP optional parameters (parameter type, length and value). Due to the way in which the SS7 stack is implemented, for protocols other than ISUP, messages are mapped to ISUP primitive and parameters format.

The following code demonstrates the use of **gc_GetSigInfo( )**:

```
char buffer[350];  // Should be enough to hold the GC/SS7 SIGINFO header
                   // + any ISUP message
S7_SIGINFO_BLK *blk_p = (S7_SIGINFO_BLK *)buffer;
gc_GetSigInfo(ldevH, buffer, U_IES, &metaevt);
// blk_p->prim is the primitive
// blk_p->length is the total dara length
// blk_p->data holds the first IE. Other IEs follow, the next one starting
//     after the end of the data for the previous one.
```

4.2.6. gc_OpenEx( )

Global Call device naming conventions apply to SS7 telephony devices. The protocol name to use is SS7. A voice device name may be specified, in which case this device will be opened and its handle will be available through the **gc_GetVoiceH( )** function. An application should use the following device name format:

N_*network_device_name*:P_SS7:V_*voice_device_name*

See the *Global Call API Library Reference* for more on the device name format.

If the circuit resides on a DM3 board, the **devicename** string should include the additional :L_SS7 component. This may also be used for other network interfaces controlled by Global Call SS7 but **must** be specified for DM3 network interfaces. The format of the **devicename** string therefore becomes:

:L_SS7:N_*network_device_name*:P_SS7:V_*voice_device_name*

The result of specifying a voice device name in the Global Call device name given to **gc_OpenEx( )** is the same as opening the voice device separately, using **dx_open( )**, and then performing a **gc_Attach( )**. This means that a voice device opened as part of a Global Call line device can later be detached from the line device using **gc_Detach( )**. A voice device that has been opened together with a Global Call line device but that has later been detached from it is not closed during the corresponding **gc_Close( )**.

The network device that is specified is the physical time slot where the voice circuit is located. This is completely independent of the signaling path. The latter need only be specified in the configuration of the system. The circuit time slot can reside

on a Intel® Dialogic network interface board (D/240SC-T1, D/300SC-E1, D/480SC-2T, and so on) or on a PCCS6 board.

For an Intel® Dialogic interface board, the standard device names are used: **dtiB*xTy*** where *x* is the logical board number and *y* is the logical circuit number (from 1 to the number of circuit on the trunk, no gaps are left for unused time slots or time slots used for signaling).

For a PCCS6 board, the devices names used are: **dkB*xTy*** where *x* is 1 for the first trunk of the board and 2 for the second trunk (if present) and *y* is the logical circuit number (same as for DTI boards).

NOTE: In this release of the software, trunk device (for example,dtiB1) may not be opened for SS7.

As part of executing **gc_OpenEx( )**, Global Call SS7 will start initializing the circuit. The application must wait for a **GCEV_UNBLOCKED** event to be received before it can start using the opened line device.

4.2.7. gc_ResetLineDev( )

The **gc_ResetLineDev( )** function releases any resource allocated to the circuit and any of its associated calls and performs a reset of the telephony circuit.

This function also cancels **gc_WaitCall( )** and sets the channel state to GCLS_INSERVICE. See Section 4.2.9, "gc_SetChanState( )", on page 57 for more information.

A GCEV_RESETLINEDEV event indicates successful completion of the function. Upon reception of this event, the application may issue a new **gc_WaitCall()** in order to start receiving calls again.

4.2.8. gc_ResultValue( )

The call control library-specific result value will indicate the actual SS7 network cause value, if available.

NOTE: The **gc_ResultValue( )** function is deprecated. The preferred alternative is **gc_ResultInfo( )**.

4.2.9. gc_SetChanState( )

The **gc_SetChanState( )** function allows an application to block a circuit. This release of Global Call SS7 will always perform maintenance blocking, whether the specified state is GCLS_MAINTENANCE or GCLS_OUT_OF_SERVICE. Consequently, any active call on the circuit will always proceed unaffected but further calls will be blocked. Setting the channel state to GCLS_INSERVICE unblocks the circuit.

4.2.10. gc_SetParm( )

The **gc_SetParm( )** function can be used to configure the following line device parameters:

- **GCPR_CALLINGPARTY** - The default calling party address for the circuit. This parameter is overwritten by the one in the S7_MAKECALL_BLK if specified. Use the paddress field of the GC_PARM union.
- **GCPR_MINDIGITS** - The minimum number of digits to collect before reporting an OFFERED call. An overlap receive procedure is used in case the initial number of digits does not reach the minimum number set using this function. Use the intvalue field of the GC_PARM union.
- **GCPR_RECEIVE_INFO_BUF** - The size of the cyclic buffers used to store signaling messages that can be retrieved using the **gc_GetSigInfo( )** function. See Section 4.2.5, “gc_GetSigInfo( )”, on page 53 for details. Use the intvalue field of the GC_PARM union.
- **GCPR_UNKNOWN_ISUP_MSGS** - Enables the configuration of a line device to receive ISUP messages not recognized by the SS7 call control library. See Section 5.12, “Sending and Receiving ISUP Messages”, on page 79 for more information.

4.2.11. gc_StartTrace( )

The **gc_StartTrace( )** function starts SS7 call control library tracing. See Section 6.3.1, “SS7 Call Control Library Trace”, on page 88 for more information. Starting a trace on one channel starts a process-wide tracing, that is, tracing on all circuits opened within the process in which **gc_StartTrace( )** was called. The function must be called on a circuit line device.

4.2.12. gc_SetConfigData( )

The **gc_SetConfigData**( ) function is used to support circuit group blocking, unblocking and resetting. The target type is GCTGT_CCLIB_SYSTEM and the target ID value is 6. The function uses the **target_datap** parameter of type GC_PARM_BLK to provide the detailed configuration information. See Section 5.11, "Circuit Group Blocking, Unblocking and Resetting", on page 77 for more information.

4.2.13. gc_SndMsg( )

The **gc_SndMsg**( ) function enables sending of application-ISUP messages, as long as they do not alter the call state or circuit state.

Messages must be formatted as required by the SS7 stack. This format is very similar to the ISUP format with the exception that all message parameters are coded as optional parameters (parameter name, length and contents).

The ISUP message type (also know as **primitive**) is specified in the *msg_type* argument. The message parameters are specified in the S7_IE_BLK pointed to by the cclib field of the GC_IE_BLK given as an argument to this function. Multiple parameters can be put one after the other in the data field of the S7_IE_BLK structure. The total length of the parameters section must be set in the length field of the structure.

The following code fragment illustrates the use of **gc_SndMsg**( ) for SS7:

```
/* Send a Subsequent Address Message
 * (SAM) with digits 234 (overlap sending)
 */
S7_IE_BLK ie_blk;
GC_IE_BLK gc_ie_blk;

ie_blk.length = 5;
ie_blk.data[0] = 0x05;  /* Parameter 1 name - Subsequent Number */
ie_blk.data[1] = 0x03;  /* Parameter 1 length - 3 bytes */
ie_blk.data[2] = 0x80;  /* Parameter 1 value - odd number of digits */
ie_blk.data[3] = 0x32;  /* Parameter 1 value - digits '2' and '3' */
ie_blk.data[4] = 0x04;  /* Parameter 1 value - digit  '4' */

gc_ie_blk.gclib = NULL;
gc_ie_blk.cclib = &ie_blk;
ret = gc_SndMsg(linedev, crn, 0x02 /* SAM */, &gc_ie_blk);
```

NOTE: Parameter values (for example, 0x05 which corresponds to the Subsequent Number parameter) should correspond to parameter values from the ISUP/TUP specifications. Similarly, message type values (for example, 0x02 in the **gc_SndMsg( )** function call above) should correspond to message type values from the ISUP/TUP specification.

4.2.14. gc_StopTrace( )

The **gc_StopTrace( )** function stops the process-wide tracing associated with a specific channel. See Section 6.3.1, “SS7 Call Control Library Trace”, on page 88. The function must be called on a circuit line device.

4.3. Call Related Functions

The SS7 call control library supports the call-related Global Call functions with the additional functionality or specific aspects described in the following subsections.

4.3.1. gc_AcceptCall( )

The **gc_AcceptCall( )** function is used to send an Address Complete Message (ACM). The **rings** parameter is ignored.

4.3.2. gc_AnswerCall( )

The **gc_AnswerCall( )** function is used to send an Answer Message (ANM). In the case of ITU-T operation, if no ACM message has been sent, the **gc_AnswerCall( )** function sends a Connect message (CON) instead of an ANM message. The **rings** parameter is ignored.

4.3.3. gc_CallAck( )

The GCST_GETMOREINFO and GCST_SENDMOREINFO states must be enabled by issuing the **gc_SetConfigData( )** function with a **target_type** of GCTGT_GCLIB_CHAN and a **target_ID** of a line device, and passing the appropriate set ID and parameter ID as follows:

- GCSET_CALLSTATE_MSK

- GCACT_ADDMSK with the following values:
  - GCMSK_GETMOREINFO_STATE
  - GCMSK_SENDMOREINFO_STATE

See the **gc_SetConfigData( )** function description in the *Global Call API Library Reference* and the section on Call State Configuration in the *Global Call API Programming Guide* for more information.

4.3.4. gc_DropCall( )

The **gc_DropCall( )** function sends a Release message (REL) to the SS7 stack if the active call has not been released by the other side. The REL message contains an SS7 cause translated from a GC cause specified as an argument to the **gc_DropCall( )** function. Otherwise, the **gc_DropCall( )** function sends a Release Complete message (RLC).

4.3.5. gc_GetCallInfo( )

The **gc_GetCallInfo( )** function can retrieve the following information:

- **CATEGORY_DIGIT** - The calling party category for the call.
- **DESTINATION_ADDRESS** - The destination address. This method of retrieving the destination address is preferred over the equivalent **gc_GetDNIS( )** function.
- **ORIGINATION_ADDRESS** - The origination address. This method of retrieving the origination address is preferred over the equivalent **gc_GetANI( )** function.
- **PRESENT_RESTRICT** - The calling party presentation restriction.
- **REDIRECTING_NUMBER** - The destination address before the last redirection (forward or diversion).

Other info_id values are not currently supported for SS7. The functionality of the U_IES (Unformatted Information Elements) info_id can be obtained by using the more appropriate **gc_GetSigInfo( )** functions that associates messages with Global Call events. See Section 4.2.5, "gc_GetSigInfo( )", on page 53 for more details.

4.3.6. gc_GetDNIS( )

The **gc_GetDNIS( )** function returns the full DNIS string available, including any digits received in overlap mode after the Initial Address Message (IAM).

NOTE: The **gc_GetDNIS( )** function is deprecated; use **gc_GetCallInfo( )**.

4.3.7. gc_HoldCall( )

At any time after a call is in the Connected state, the application can call the **gc_HoldCall( )** function to put the call in the Suspended state. The application receives a GCEV_HOLDACK event indicating that the call has entered the Suspended state. The call remains in the Suspended state until the **gc_RetrieveCall( )** function is called with the same CRN to resume the call. See Section 4.3.9, “gc_RetrieveCall( )”, on page 62 for related information.

4.3.8. gc_MakeCall( )

The SS7 call control library supports the timeout parameter, even in asynchronous mode.

The GC_MAKECALL_BLK data structure contains a **cclib** field. When the **cclib** field is set to zero, default values are used for all call setup parameters. When the **cclib** field is set to a pointer to an S7_MAKECALL_BLK data structure which contains parameters usually set in an Initial Address Message (IAM), the specified fields overwrite the default values in the IAM.

The S7_MAKECALL_BLK structure contains the following IAM parameters:

- destination_number_type
- destination_number_plan
- internal_network_number
- origination_phone_number
- origination_number_type
- oringination_number_plan
- calling_party_category
- origination_present_restrict

- origination_screening
- forward_call_indicators
- trans_medium_req
- satellite_indicator
- echo_device_indicator
- continuity_check_indicator

NOTE: Other parameters can be added using the **gc_SetInfoElem( )** function. See Section 4.3.11, "gc_SetInfoElem( )", on page 63 for more information.

The **gc_MakeCall( )** function can be used to request an in-call continuity check. The continuity_check_indicator in the S7_MAKECALL_BLK structure must be set to CCI_CC_REQUIRED so that Global Call will send an SS7 IAM message with continuity check to the network. See Section 5.10.2, "Outbound Continuity Check", on page 75 for more information.

4.3.9. gc_RetrieveCall( )

An application can use the **gc_RetrieveCall( )** function to resume a call previously placed in the Suspended state by using the **gc_HoldCall( )** function. The application receives a GCEV_RETRIEVEACK event if the call is resumed successfully. If the network has placed the call in the Suspended state, a call to **gc_RetrieveCall( )** to resume the call will fail. See Section 4.3.7, "gc_HoldCall( )", on page 61 for related information.

4.3.10. gc_SetBilling( )

The **gc_SetBilling( )** function may be used before calling **gc_AcceptCall( )** or **gc_AnswerCall( )** to control charging (charge or no-charge). After the **gc_SetBilling( )** function is called, Global Call sets accordingly the BCI (Backward Call Indicator) parameter in each ACM or CON message that it sends.

- If the specified rate type is any value other than GCR_NOCHARGE, the charge indicator of the BCI is set to **charge**.
- If the specified rate type is GCR_NOCHARGE, the charge indicator of the BCI is set to **no charge**.

The charge indicator is left in the default value in case the **gc_SetBilling( )** function is not called by the application.

4.3.11. gc_SetInfoElem( )

The **gc_SetInfoElem( )** function allows the application to add ISUP message parameters (that is, information elements) to outgoing messages sent by the SS7 call control library while executing a Global Call call control function. The format of the information elements is typically identical to the ISUP format, with the exception that all parameters are formatted as optional parameter (parameter name, length, and contents). It is possible to add multiple information elements in one **gc_SetInfoElem( )** function call. The parameters must be put in an S7_IE_BLK structure, a pointer to which is set in the cclib field of the GC_IE_BLK specified as argument to the function. The following code fragment illustrates the use of the function:

```
/*  Add User-to-user information to Initial Address Message */
S7_IE_BLK ie_blk;
GC_IE_BLK gc_ie_blk;

ie_blk.length = 5;
ie_blk.data[0] = 0x20;  /* Parameter name - User-to-user info */
ie_blk.data[1] = 0x03;  /* Parameter length - 3 bytes */
ie_blk.data[2] = 'A';   /* Parameter value - 1st byte */
ie_blk.data[3] = 'B';   /* Parameter value - 2nd byte */
ie_blk.data[4] = 'C';   /* Parameter value - 3rd byte */

gc_ie_blk.gclib = NULL;
gc_ie_blk.cclib = &ie_blk;
if (gc_SetInfoElem(linedev, &gc_ie_blk) != GC_SUCCESS) /* Process error */
if (gc_MakeCall(linedev, &crn, "7124311", NULL, 15, EV_ASYNC) != GC_SUCCESS)
/* Process error */
```

NOTE: Parameter values (such as 0x20 in the example above, which corresponds to the User-to-User Information parameter) should correspond to parameter values from the ISUP/TUP specifications.

5. SS7-Specific Tasks

5.1. Overview

This chapter describes how Global Call is used to handle certain SS7-Specific tasks. These tasks include:

- Handling of Glare Conditions
- Controlling Priority in Circuit Groups
- Routing
- Connecting Multiple Hosts to SIUs
- Benefits of Dual Resilient SIU Configurations
- Configuration of Dual Resilient SIUs
- Overlap Send and Receive
- Call Suspend and Resume
- Continuity Check
- Circuit Group Blocking, Unblocking and Resetting
- Sending and Receiving ISUP Messages

5.2. Handling of Glare Conditions

A glare condition occurs when an outgoing call has been initiated (**gc_MakeCall( )** succeeded) and an incoming call is detected. Global Call SS7 and the SS7 stack almost completely hide this condition from the application that will see its outbound call fail and will then be notified of the inbound call.

However, in order to avoid adding delay to the handling of the inbound call, the SS7 call control library does not wait for the failed outbound call to be released before it notifies the application of the inbound call. This means that, in case of glare, the following type of scenario can be seen:

Application	Libgcs7
gc_MakeCall(crn1) -->	
	GCEV_DISCONNECTED(**crn1**) <--
gc_DropCall(crn1) -->	
	GCEV_OFFERED(**crn2**) <--
gc_AcceptCall(crn2) -->	
	GCEV_DROPCALL(**crn1**) <--
gc_ReleaseCallEx(crn1) -->	

This shows that an application running on bidirectional circuits should be ready to handle two CRNs on a single line device. However, the application can be purely "reactive" with respect to the failed call (crn1) and just respond to events using their associated CRN: simply perform a **gc_ReleaseCallEx( )** upon reception of any GCEV_DROPCALL, whether the CRN is the "active" one or not. Using this procedure, the application only needs to store one CRN per line device.

Another case of glare condition is at disconnection. If the application calls **gc_DropCall( )** while a GCEV_DISCONNECTED has already been put in the SRL event queue, the application will receive it after it does **gc_DropCall( )** when it is waiting for GCEV_DROPCALL. This late GCEV_DISCONNECTED event must be ignored by the application. The call control library will send the GCEV_DROPCALL as usual when the call is dropped. Other glare conditions at disconnection are all hidden from the application.

5.3. Controlling Priority in Circuit Groups

ISUP allows the setting of different priority schemes on a per circuit group basis:

- Priority to incoming call on all circuits
- Priority to outgoing call on all circuits
- Highest point code has priority on even CICs (Circuit Identification Codes)
- Highest point code has priority on odd CICs

The third scheme is the one recommended by the ITU (Q.764). With the SS7 stack, the priority scheme can be selected in the **<options>** field of the **ISUP_CFG_CCTGRP** commands in the *config.txt* file. Once priority has been given to one of the calls by the SS7 stack, upper software layers (Global Call SS7 and the application) must conform.

Because of the multiple layers of the software architecture and the asynchronous nature of the communication between them, it is possible that collisions appear to exist even though there has not been a true glare condition on the signaling link. For example, if the SS7 stack has posted an IAM message for the GlobaCall SS7 call control library but that the application issues a **gc_MakeCall( )** before this message is received, the application will see the equivalent of a glare condition: the outbound call will fail and the inbound call will be offered. This can happen regardless of the configured priority scheme, even with priority given to outbound calls on all circuits.

5.4. Routing

Routing is described under the following topics:

- Routing Functions
- Time Slot Assignment for Intel® NetStructure™ SS7 Boards
- Using Time Slot 16 on Intel® Dialogic E-1 Network Interface Boards

5.4.1. Routing Functions

The Global Call SS7 Call Control Library (Libgcs7) supports the Global Call routing functions (**gc_Listen( )**, **gc_UnListen( )**, and **gc_GetXmitSlot( )**). These

functions are available to user application for performing routing of SS7 circuits regardless of their physical location (for example, on an Intel® Dialogic network interface (DTI) board or on an Intel® NetStructure™ SS7 board). This allows the application to use one single set of functions without having to know where the circuit is located (that is, on a DTI board or on an Intel® NetStructure™ SS7 board).

The following functions are provided:

- **int gc_Listen(LINEDEV linedev, SC_TSINFO *sctsinfo_p, mode)**
- **int gc_Unlisten(LINEDEV linedev, mode)**
- **int gc_GetXmitSlot(LINEDEV linedev, SC_TSINFO *sctsinfo_p, mode)**

5.4.2. Time Slot Assignment for Intel® NetStructure™ SS7 Boards

The SS7 server automatically assigns CT Bus transmit time slots for telephony circuits located on an Intel® NetStructure™ SS7 board. The SS7 server also performs the full-duplex routing required for the signaling connection, when the signaling links are routed over the CT Bus between an Intel® NetStructure™ SS7 board and a network interface (DTI) board. The configuration required for this to happen is described in Section 3.3.1, "TDM Bus Configuration of an Intel® NetStructure™ SS7 Board", on page 29.

In both cases, the number of time slots used by the an Intel® NetStructure™ SS7 board to transmit data and signaling over the CT Bus must be reserved using the Third Party Tech option in DCM as described in Section 3.4.1, "Using the Third Party Tech Option in DCM to Manage the CT Bus", on page 35. The Intel® Dialogic SS7 server checks that a sufficient number of timeslots has been specified. If not enough timeslots are reserved, the SS7 server will not start and will generate an error.

5.4.3. Using Time Slot 16 on Intel® Dialogic E-1 Network Interface Boards

Traditionally, E-1 trunks reserve physical time slot 16 for signaling, which is designated as dtiB#T31, where # is the logical number of the trunk. With SS7 however, signaling is on a different physical trunk than the telephony circuits. The signalling time slots can then be used for a normal voice circuit.

With Intel® Dialogic E-1 network interface boards, setting time slot 16 to the “clear channel” mode requires that special ISDN firmware to be downloaded to the board and that ISDN D channel be disabled. For Springware boards, this can be done using the CTR4 (ISCTR4 v6.65) firmware for example and by changing parameter 16 in the *CTR4.PRM* parameter file to 2. For DM3 boards, the special _TS16 firmware should be used.

Similarly, if an SS7 link is routed from time slot 16 of an Intel® Dialogic E-1 network interface board to an Intel® NetStructure™ SS7 board, the Intel® Dialogic board must leave time slot 16 in **clear channel** mode, as described.

5.5. Connecting Multiple Hosts to SIUs

SIU systems may have multiple hosts connected to the same SIU or pair of SIUs. In this case, each host is responsible for the telephony circuits that it terminates. This must be specified in the *config.txt* file on the SIU(s). Each ISUP_CFG_CCTGRP command must specify in its **<host_id>** field which host is responsible for the circuit group. Additionally, the *config.txt* file must also specify, using the SIU_HOSTS command, the number of hosts that will be used.

On each host, the **HostID** parameter must be set to reflect which one is the local host. This allows Global Call SS7 to correctly identify the host when communicating with the SIU(s) and to know which circuit groups are configured on the local host.

5.6. Benefits of Dual Resilient SIU Configurations

A dual-resilient SIU configuration brings an additional level of fault tolerance to a Global Call SS7 system. It consists of two SIUs configured as a single point code in the SS7 network. Host systems are connected via TCP/IP to both servers.

Under normal circumstances (both servers up and running) the load is shared between both units (see Section 3.3.3, “ISUP Configuration”, on page 32). If one unit fails - either the whole unit or its communication with the hosts -, the partner unit maintains MTP operation of the node. However, telephony circuit groups that were active on the failing SIU need to be transferred to the partner SIU in order to be restored. With Global Call SS7, this procedure is performed automatically by the

Intel® Dialogic SS7 server. The application will only see that circuits are blocked (GCEV_BLOCKED event is received) and then unblocked after they are successfully transferred to the partner SIU. The application should handle this as any other case of blocked circuits.

Global Call SS7 automatically handles the restoration of the circuit groups to their "preferred" SIU when it comes back up after a failure. Again, the only thing the application will notice is that circuits are blocked before they are transferred and unblocked when the transfer is complete. Because this transfer must be done for a complete circuit group, Global Call SS7 will block each circuit in the group as they become idle. Circuits that have an active call are only blocked after the call is finished. Once all circuits are blocked, the transfer to the preferred SIU is performed and circuits are then unblocked.

5.7. Configuration of Dual Resilient SIUs

Dual-resilient SIU systems must have two SIUs configured. For Windows systems, this configuration is done in the Intel® Dialogic Configuration Manager (DCM). For Linux systems, this configuration is done in the *ss7.cfg* file. SIUs are configured as either SIU A or SIU B. The first SIU configured must be SIU A, and the second SIU must be B.

For Global Call SS7 to be able to automatically handle dual-resilient SIU operations, the *config.txt* file must specify which is the preferred SIU for each circuit group. See Appendix C, "Sample Configuration Files" for an examples of the configuration files.

5.8. Overlap Send and Receive

The S77 call control library supports overlap sending using the **gc_SendMoreInfo( )** function.

Two methods of overlap receiving are supported, the preferred method, and an older method maintained for backward compatibility reasons only. Both methods are described below.

The preferred method for implementing overlap receiving is as follows:

1. Issue **gc_CallAck(GCACK_SERVICE_INFO)** to determine if digits are available.
2. Receive a GCEV_MOREINFO event.
3. Use **gc_ResultValue( )** to determine the status, which is one of the following:
   - GCRV_INFO_PRESENT_ALL - The requested digits are now available.
   - GCRV_INFO_PRESENT_MORE - The requested digits are now available. More/additional digits are available.
   - GCRV_INFO_SOME_TIMEOUT - Only some digits are available due to a time out.
   - GCRV_INFO_SOME_NOMORE - Only some digits are available, no more digits will be received.
   - GCRV_INFO_NONE_TIMEOUT - No digits are available due to a time out.
   - GCRV_INFO_NONE_NOMORE - No more digits are available.
4. Issue **gc_GetCallInfo(DESTINATION_ADDRESS)** to retrieve the digits.
5. If the status returned via GCEV_MOREINFO in step 3 indicates that more digits are available, the application can do the following:
   - Issue **gc_ReqMoreInfo( )** to request the additional digits.
   - Receive a GCEV_MOREINFO event with a status as indicated in step 3 above.
   - Issue **gc_GetCallInfo(DESTINATION_ADDRESS)** to retrieve the additional digits.
6. Repeat step 5 until all information has been retrieved.

The following method of overlap receiving continues to be supported for backward compatibility reasons only:

1. Issue **gc_CallAck(GCACK_SERVICE_DNIS)** identifying the number of digits to retrieve (dnis.accept) in the GC_CALLACK_BLK structure pointed to by the **callack_blkp** function parameter.
2. Receive a GCEV_MOREDIGITS event.
3. Issue **gc_GetDNIS( )** to retrieve the digits.

NOTE: To retrieve a certain number of digits at a time, specify that number in the dnis.accept field and repeat steps 1, 2 and 3 above until all information has been retrieved.

See the *Global Call API Programming Guide* for more detailed information on overlap sending and receiving in general and the *Global Call API Library Reference* for more information about the functions mentioned above.

5.9. Call Suspend and Resume

Call suspend and resume features are supported using the **gc_HoldCall( )** and **gc_RetrieveCall( )** functions. A call can be suspended by the application or by the network.

When a call is in the Connected state, the application can issue **gc_HoldCall( )** on the CRN of the current call to put the call in the suspended state. The application receives a GCEV_HOLDACK event indicating that the call has entered the suspended state. The call remains in the suspended state until a **gc_RetrieveCall( )** is issued on the CRN for the call. The application receives a GCEV_RETRIEVEACK event when this occurs.

If the action of suspending a call is initiated by the network (with an SS7 SUS message), the application receives a GCEV_HOLDCALL event. When the network resumes the call, the application receives a GCEV_RETRIEVECALL event. If the network decides to drop the call or the call remains in the suspended state for too long, the application will not receive the GCEV_RETRIEVECALL event but instead receives a GCEV_DROPCALL event. While a call is in the suspended state, it can be dropped or released by the application.

NOTES:

1. The Global Call call state, as returned by **gc_GetCallState( )**, for a suspended call is GCST_ONHOLD.
2. A suspended call can only be resumed by the side that originally put the call in the suspended state. If a call has been placed in the suspended state by the network, the application **cannot** resume the call using the **gc_RetrieveCall( )** function. The **gc_RetrieveCall( )** function will fail if this is attempted. Similarly, if a call has been placed in the suspended state by the application, an SS7 RES message from the network will **not** resume the call.

5.10. Continuity Check

The continuity check feature is implemented using the **gc_Extension( )** function and the associated GCEV_EXTENSION event.

The structure associated with the GCEV_EXTENSION event (METAEVENT structure) contains the extevtdatap field which is a pointer to an EXTENSIONEVTBLK structure. The value of the ext_id field in the EXTENSIONEVTBLK structure can be:

- S7_EXT_CONTCHECK to indicate the beginning of a continuity check process
- S7_EXT_CONTCHECK_END to signal the end of a continuity check process

The parmblk field in the EXTENSIONEVTBLK structure contains additional information. The parmblk field, which is of type GC_PARM_BLK, contains only one element of parameter data of type GC_PARM_DATA. The set ID of this parameter is S7SET_CONTCHECK and the parameter ID is S7PARM_CONTCHECK_TYPE. The parm_data_size is sizeof(int).

In this feature, the **gc_Extension( )** function does not require any GC_PARM_BLK data, except when sending continuity check result and the outcome of the test must be sent. Also, the **gc_Extension( )** function does not return anything via the **retblkp** parameter.

5.10.1. Inbound Continuity Check

When a continuity check request is received from the network, the GCEV_EXTENSION event is sent to the application with the value of the S7PARM_CONTCHECK_TYPE parameter (as described in Section 5.10, “Continuity Check”, on page 73) set to S7RV_CC_INBOUND. Upon receipt of this event, the application must do the following:

- Save, if necessary, the current time slot assignment of the current line
- Mark the current line as unavailable for outgoing calls until a GCEV_EXTENSION event is received with the ext_id field set to S7_EXT_CONTCHECK_END
- Put the line in loopback for the continuity test by calling **gc_Extension( )** with an **ext_id** value of S7_EXT_APPLYLOOPBACK

NOTE: If the line is not put in loopback mode, the continuity check will always fail. The **gc_Extension( )** function with the **ext_id** set to S7_EXT_APPLYLOOPBACK puts the line in loopback mode for the purpose of running the continuity check. In ANSI networks, an SS7 LPA message is sent to the network.

When the continuity check process is completed, the application receives a GCEV_EXTENSION event with an **ext_id** of S7_EXT_CONTCHECK_END.

NOTE: The **gc_Extension(S7_EXT_APPLY_LOOPBACK)** function changes the timeslot that the network device is listening to. Therefore, when the continuity test finishes, the application must use the **gc_Listen( )** function to restore the timeslot that is being transmitted out of the network interface (that is, the CT Bus timeslot that the network device is listening to).

For the inbound continuity check, the S7PARM_CONTCHECK_TYPE parameter value can be S7RV_CCEND_INBOUND_SUCCESS or S7RV_CCEND_INBOUND_FAILURE. If the continuity check fails (S7RV_CCEND_INBOUND_FAILURE), the application may decide not to use the circuit until a new continuity check completes successfully on that circuit.

NOTE: In some cases, multiple GCEV_EXTENSION events with an ext_id of S7_EXT_CONTCHECK may arrive on the same line device before a GCEV_EXTENSION event with and ext_id of S7_EXT_CONTCHECK_END is received. This is not an error but an indication that the network is re-checking the same circuit. However, the application must put the line in loopback mode every time.

For continuity check requests received during call setup (the SS7 IAM message), the same rules apply. The application will receive a GCEV_EXTENSION event with an **ext_id** equal to S7_EXT_CONTCHECK and the S7PARM_CONTCHECK_TYPE parameter value set to S7RV_CC_INBOUND. Thereafter, when the Continuity Check finishes, another GCEV_EXTENSION event with an ext_id of S7_EXT_CONTCHECK_END is received. In the latter case, success or failure of the continuity check process is also indicated by a parameter value of S7RV_CCEND_INBOUND_SUCCESS or S7RV_CCEND_INBOUND_FAILURE. If the continuity check is successful, the application receives a GCEV_OFFERED event following the GCEV_EXTENSION event. If the continuity check is not successful, the application will not be aware of the call attempt.

If an SS7 IAM message is received with a “continuity check performed on previous circuit” indication, the GCEV_OFFERED event is not sent to the application until the call control library receives an SS7 COT message indicating the success of the continuity check.

NOTE: The **gc_Extension(APPLY_LOOPBACK)** function changes the timeslot that the network device is listening to. Therefore, when the continuity test finishes, the application must use the **gc_Listen( )** function (or **dt_listen( )** for a DTI device) to restore the timeslot that is being transmitted out of the network interface (that is, the CT Bus timeslot that the network device is listening to).

5.10.2. Outbound Continuity Check

As for the inbound continuity check, the outbound continuity check can be done outside of any call (Out-of-Call) or as part of an outgoing call (In-Call). However, in the outbound case, since the check is initiated by the application, the procedures for both types of check differ.

Outbound Out-of-Call Continuity Check

When requesting an outbound **out-of-call** continuity check on a circuit, the line device must be in the Idle state, that is, the circuit must be unblocked and cannot have any active calls. The application can then use the **gc_Extension( )** function with an **ext_id** of S7_EXT_REQUESTCONTCHECK to send an SS7 CCR message to the network.

The application receives a GCEV_EXTENSION event with an ext_id of S7_EXT_CONTCHECK and with a parameter value of S7RV_CC_OUTBOUND to indicate that it can begin the continuity check by connecting the test equipment to the line.

When the continuity check is completed and the result analyzed, the application must call **gc_Extension( )** with an **ext_id** of S7_EXT_SENDCONTCHECKRESULT to communicate the results of the check to the remote party. To achieve this, the application must build a GC_PARM_BLK structure. The set_ID must be S7SET_CONTCHECK and the param_ID must be S7PARM_CONTCHECK_RESULT and the parameter value must be either CONTI_SUCCESS or CONTI_FAILURE.

If the function is called with CONTI_SUCCESS, the continuity check process is finished and the application is notified by a GCEV_EXTENSION event with an ext_id of S7_EXT_CONTCHECK_END and with a parameter value of S7RV_CCEND_OUTBOUND. When the application receives this event, the line can be used for making or receiving calls.

If the function is called with CONTI_FAILURE, the remote side is waiting for a re-check, and therefore the application does not receive a GCEV_EXTENSION event with an ext_id of S7_EXT_CONTCHECK_END.

Outbound In-Call Continuity Check

To request an **in-call** continuity check, the application must call **gc_MakeCall( )** with the continuity_check_indicator field in the S7_MAKECALL_BLK structure set to CCI_CC_REQUIRED, so that the Global Call library sends an SS7 IAM message, with continuity check requested, to the network.

The application receives a GCEV_EXTENSION event with an ext_id of S7_EXT_CONTCHECK and with a parameter value of S7RV_CC_OUTBOUND to indicate that it can begin the continuity check by connecting the test equipment to the line.

If the continuity check is successful, the application indicates the success to the remote side by calling **gc_Extension( )** with an **ext_id** of S7_EXT_SENDCONTCHECKRESULT and a parameter value of CONTI_SUCCESS. Since the continuity check process is now finished, the application receives a GCEV_EXTENSION event with an ext_id of S7_EXT_CONTCHECK_END with a parameter value of S7RV_CCEND_OUTBOUND. When the application receives this event, the call proceeds in the normal way.

If the continuity check fails, to indicate the failure to the remote side, the application must call **gc_DropCall( )** with a cause value of CONTCHECK_FAILED. The call is cleared internally by Global Call and the other side will have no knowledge of the call. The other side only recognizes a failed continuity check test and waits for a re-check.

CAUTION

If a failure result is sent to the other side, the other side will expect a re-check on the circuit. Therefore, another call to **gc_Extension( )** with an **ext_id** of S7_EXT_REQUESTCONTCHECK should be issued by the application, until the continuity check succeeds. Alternatively, the application could reset the circuit using **gc_ResetLineDev( )** on the corresponding line device. In this case, the application does not receive a GCEV_EXTENSION event, but receives a GCEV_RESETLINEDEV event corresponding to the **gc_ResetLineDev( )** function call.

The GCEV_EXTENSION event with an ext_id of S7_EXT_CONTCHECK_END may be received in two other cases:

- If the parameter value is CCEND_OUTBOUND_ERROR, an error occurred during the continuity check, for example, if the time waiting for the SS7 REL message at the remote side expires.
- If the parameter value is CCEND_OUTBOUND_GLARE, a glare condition occurred, for example, while seizing the line for a continuity check, an SS7 IAM message was received.

CAUTION

In both cases of the GCEV_EXTENSION event with ext_id of EXT_CONTCHECK_END above, the continuity check process is abandoned by the Global Call library. The application should not try to perform the physical continuity test again or try to send any continuity check results because the remote side is not ready to receive the results and the send operation will fail.

5.11. Circuit Group Blocking, Unblocking and Resetting

SS7 supports the block, unblock and reset maintenance operations at the circuit group level as well as on a per circuit basis. However, when the same operation must be performed on many circuits, sending a separate message for each individual circuit does not make efficient use of the signaling network bandwidth and may even overload some switches. Using circuit group maintenance messages is a better alternative in such cases.

The Global Call SS7 call control library will attempt to achieve this optimization transparently for the application. The application can invoke the **gc_ResetLineDev( )** or **gc_SetChanState( )** functions for individual circuits (line devices) but these circuit maintenance operations will be buffered by the library for a small amount of time, allowing more requests for other circuits within the same circuit group to be grouped with the original request and sent as one single circuit group maintenance message.

In addition, the application may elect to request circuit group maintenance operations directly on the groups through the **gc_SetConfigData( )** function. This method is not recommended for general use because it is specific to the SS7 call control library. However, if for any reason the transparent mode described in the previous paragraph is not sufficient, the application has access to the full flexibility provided by the **gc_SetConfigData( )** function. In such cases, the **gc_SetConfigData( )** function can be called with a target type of GCTGT_CCLIB_SYSTEM and a target ID of GC_SS7_LIB. The **target_datap** parameter (of type GC_PARM_BLK) should be built using **gc_util_insert_parm_ref( )** with a **setID** of S7SET_CCTGRP, a **parmID** of S7PARM_CCTGRP_STATE, and the value being the address of a S7PARM_CCTGRP_STATE_DATA structure. The S7PARM_CCTGRP_STATE_DATA structure contains the state field where the desired block, unblock, or reset operation is specified. The operation can be one of the following:

- S7_CCTGRP_STATE_MAINT_BLOCK - Put the circuits in a maintenance block state. In this state, active calls can continue, but new calls are not allowed.
- S7_CCTGRP_STATE_HW_BLOCK - Put the circuits in a hardware block state. In this state, active calls are disconnected and new calls are not allowed.
- S7_CCTGRP_STATE_MAINT_UNBLOCK - Undo a maintenance block operation.
- S7_CCTGRP_STATE_HW_UNBLOCK - Undo a hardware block operation.
- S7_CCTGRP_STATE_RESET - Reset the circuits, clearing any call and local maintenance or hardware block state.

5.12. Sending and Receiving ISUP Messages

The **gc_SndMsg( )** function can be used to send any ISUP message (for example, facility) that does not alter the call state or circuit state. See Section 4.2.13, "gc_SndMsg( )", on page 58 for more information.

Incoming ISUP messages that trigger Global Call events can be retrieved using the **gc_GetSigInfo( )** function. See Section 4.2.5, "gc_GetSigInfo( )", on page 53 for more information.

Global Call can also be used to configure a line device to receive ISUP messages processed by the underlying stack but not recognized by the SS7 call control library. To configure a line device to receive these ISUP messages, use the **gc_SetParm( )** function as follows:

```
GC_PARM t_gcparm;
t_gcparm.intvalue = true;
gc_SetParm(ldev, GCPR_UNKNOWN_ISUP_MSGS, t_gcparm);
```

When an ISUP message is received on the line device, a GCEV_EXTENSION event with an ext_id of S7_EXT_ISUP_EVENT is generated. The application can retrieve the message parameters using code similar to the following:

```
void getextevtdata(METAEVENT* a_me_p) {
    int ext_id = ((EXTENSIONEVTBLK*)(a_me_p->extevtdatap))->ext_id;
    if (S7_EXT_ISUP_EVENT == ext_id) {
        GC_PARM_BLKP t_parmblk_p =
            &(((EXTENSIONEVTBLK*)a_me_p->extevtdatap)->parmblk);
        GC_PARM_DATAP t_parm_p =
            gc_util_find_parm(t_parmblk_p, S7SET_ISUP_EVENT,
                              S7PARM_ISUP_EVENT_PARM);
        if (t_parm_p) {
            printf(" parm size=%d. 0x...", t_parm_p->value_size);
            for (int i=0; i < t_parm_p->value_size; ++i) {
                printf(" %02x", t_parm_p->value_buf[i]);
            }
        }
    }
}
```

6. Troubleshooting

6.1. Proving the Configuration

An important step in troubleshooting a Global Call SS7 system is proving the SS7 stack configuration and the SS7 network connection (links), independently of any Intel® Dialogic component.

6.1.1. Intel® NetStructure™ SS7 Board Systems

Verify an Intel® NetStructure™ SS7 board system configuration as follows:

1. Depending on the type of Intel® NetStructure™ SS7 board being used, add debug flags to the *system.txt* file as follows:
   - For Intel® NetStructure™ SS7 ISA (PCCS6) board systems: Add debugging flags to SSD and S7_MGT modules and make sure S7_LOG is launched:

     ```
     FORK_PROCESS    .\SSD.EXE -d
     FORK_PROCESS    .\SSD_POLL.EXE
     FORK_PROCESS    .\TIM_NT.EXE
     FORK_PROCESS    .\TICK_NT.EXE
     FORK_PROCESS    .\S7_MGT.EXE -d
     FORK_PROCESS    .\S7_LOG.EXE -m0xef
     ```

   - For Intel® NetStructure™ SS7 CompactPCI and PCI board systems: Add debugging flags to SSD and S7_MGT modules and make sure S7_LOG is launched:

     ```
     FORK_PROCESS    .\SSD.EXE -d
     FORK_PROCESS    .\TIM_NT.EXE
     FORK_PROCESS    .\TICK_NT.EXE
     FORK_PROCESS    .\S7_MGT.EXE -d
     FORK_PROCESS    .\S7_LOG.EXE -m0xef
     ```

2. Start GCTLOAD and watch out for any error message (for example, “Timeout waiting for…”). A first part of the boot sequence should show messages similar to the following (important steps are in bold):

```
(61)gctload: Initialisation complete

S7_log : mod ID=0xef, options=0xaf0d
S7MGT >> M-t7f0f-i0000-fcf-dcf-s00-p(8)00ff000000000000
S7MGT << M-t7f0f-i0000-fcf-dcf-s00-p(8)00ff000000000000
S7MGT >> M-t7680-i0000-fcf-d20-s00-p(24)200000cf70637337332e646331000000000
0000000000010
ssd : 16 boards
S7MGT << M-t3680-i0000-fcf-dcf-s00-p(24)200000cf70637337332e646331000000000
0000000000010
S7MGT >> M-t7681-i0000-fcf-d20-s00-p(24)000100000000697375703736 2e646332000
0000000000000
ssd[0] : pccs6
ssd[0] : reset requested
S7MGT << M-t3681-i0000-fcf-dcf-s00-p(24)000100000000697375703736 2e646332000
0000000000000
ssd[0] : code download requested
ssd[0] : code download started(isup76.dc2)
ssd[0] : code download complete
ssd[0] : run requested
ssd[0] : running
```

If a "Timer expiry" message is displayed after the "ssd[0] : reset requested" line above, it is likely that the I/O settings are incorrect. Try different I/O port and/or SRAM address settings. These settings can be changed using the PCCSXCFG tool as described in the *SS7 Programmer's Manual for PCCS6*. Remember that the I/O port value must also be changed accordingly on the SW1 switch on the board.

If a "Reset failed" message is displayed on the console, check that the PCCS device is started. Open the "Devices" part of the Windows NT Control Panel and check the status of the PCCS device. The PCCS device is started when you run the PCCSXCFG toll. However, the PCCS device is set to manual mode and is consequently not restarted when you reboot your system. It is recommended to change the startup mode to Automatic.

For Intel® NetStructure™ CompactPCI and PCI boards, there should be an Intel® NetStructure™ SS7 device. In Windows 2000, right click on **My Computer**, then choose **Manage** to open the **Computer Managerment** dialog. Select **Device Manager**, then choose **Show hidden devices** from the **View** menu. Under **Non-Plug and Play Drivers**, there should be an Intel® NetStructure™ SS7 device and you can check if it has started or not.

NOTE: If an Intel® NetStructure™ SS7 device does not appear in the **Non-Plug and Play Drivers** list, type `net start Septel` at the command line.

At this point, the last thing you should see on the console is "**S7_MGT Boot Complete**" and a final S7_MGT message:

```
S7MGT << M-t3311-i0001-fcf-dcf-s00-p(16)00010171000100060000000000000000
S7MGT >> M-t7312-i0000-fcf-d22-s00-p(32)000000020000000000000002000000000000
00000000000000000000000000000000
S7MGT << M-t3312-i0000-fcf-dcf-s00-p(32)000000020000000000000002000000000000
00000000000000000000000000000000
S7MGT >> M-t7700-i0000-fcf-d23-s00-p(40)0475234d4d22004d0001000a00020040011
085000000000000000000000000000000000000000000000
S7MGT << M-t3700-i0000-fcf-dcf-s00-p(40)0475234d4d22004d0001000a00020040011
085000000000000000000000000000000000000000000000
S7MGT >> M-t7701-i0000-fcf-d23-s00-p(30)000000010000000200010001 7fff7fff001e
000000000000000000000000
S7MGT << M-t3701-i0000-fcf-dcf-s00-p(30)000000010000000200010001 7fff7fff001e
000000000000000000000000
S7MGT >> M-t7701-i0001-fcf-d23-s00-p(30)000000010000000200210021 7fff7fff001e

000000000000000000000000
S7MGT << M-t3701-i0001-fcf-dcf-s00-p(30)000000010000000200210021 7fff7fff001e
000000000000000000000000

        ****************************
        *   S7_MGT Boot complete   *
        ****************************

S7MGT << M-t3f0f-i0000-fcf-dcf-s00-p(8)00ff000000000000
```

3. Check that all messages on this last screen have a status of 0, indicating success. The status is indicated as `-sXX` in each message, where XX is an hexadecimal number.Does this mean that the successful status is -s00? This is confusing.
4. If, instead of the "S7_MGT Boot complete" message, you receive an "S7_MGT: Timeout occurred" message, check that the firmware you have specified in your *config.txt* file corresponds to the license button installed on the board.
5. Activate the MTP link(s) using the MTPSL tool (assuming linkset id 0 and link 0):

   ```
   MTPSL ACT 0 0
   ```

   If the MTP link is configured properly and activated at the adjacent point code and your system is properly clocked (this might require that Intel® Dialogic

boards be downloaded if the Intel® NetStructure™ SS7 ISA (PCCS6) board is taking its clock from the SCbus), you should see messages similar to the ones below. The important thing to check for is the presence of "Destination available".

```
S7L:I00 MTP Event : linkset_id/link_ref=0000 Changeback
S7L:I00 MTP Event : linkset_id=00 Link set recovered
S7L:I00 MTP Event : linkset_id=00 Adjacent SP accessible
S7L:I00 MTP Event : point code=00000002
 Destination available
```

If no other message appears on the console after a couple of minutesFred thinks the phrase "for a couple more minutes" is inappropriately informal as well as being vague., you can reasonably assume that your configuration is correct.

6.1.2. SIU Systems

For proving the configuration of an SIU based system, follow the steps described in this section. This description assumes a single host system connected to a single SIU.

1. Check that your *system.txt* file on the host system contains all standard LOCAL module definitions and the following FORK_PROCESS commands:

   ```
   FORK_PROCESS    .\S7_LOG.EXE -m0xef
   FORK_PROCESS    .\RSI.EXE -r.\RSI_LNK.EXE -l1
   ```

2. Run GCTLOAD and power up the SIU.
3. Establish the TCP/IP link with the SIU using the following command (where <SIU_IP_Address> is the actual IP address assigned to the SIU):

   ```
   RSI_CMD 0 0xef 0 <SIU_IP_Address> 9000
   ```

   When the SIU is booted, you should see the following messages on the S7_LOG screen (where GCTLOAD is running):

   ```
   S7L:I00 RSI_MSG_LNK_STATUS : Link 0 now down
   S7L:I00 RSI_MSG_LNK_STATUS : Link 0 now up
   ```

   The second message indicates that the host system is able to communicate with the SIU. If the link remains down, check that all LEDs on the SIU are lit. Also check the IP address of the SIU doing a **ping** to it. If not all the LEDs are lit before establishing the TCP/IP link, it may indicate a mistake in the configuration of the SIU (*config.txt* or system settings) or a hardware problem.

See the *DSC131/DSC231 User's Manual* for more information on diagnosing and solving such problems.

Once the TCP/IP link between the host system and the SIU is established, the SIU will start activating its MTP links. Messages similar to the following ones should appear on the console:

```
S7L:I00 Level 2 State : id=0 INITIAL ALIGNMENT
S7L:I00 Level 2 State : id=0 ALIGNED READY
S7L:I00 Level 2 State : id=0 IN SERVICE
S7L:I00 MTP Event : linkset_id/link_ref=0000 Changeback
S7L:I00 MTP Resume, dpc=00000001
```

The last message indicates that the destination point code (00000001 in this example) is reachable. If you do not see this and the link is activated at the adjacent point code, check the *config.txt* file on the SIU. Start by checking the point codes, the Signaling Link Code (SLC) and Sub-Service Field (SSF) parameters.

6.2. Common Problems and Solutions

The following paragraphs list mistakes that are often made while installing and configuring a Global Call SS7 system. The symptoms are described together with suggested approach to fix the problem.

6.2.1. Intel® Dialogic SS7 Server Fails to Start

In Windows systems, view the system log using the NT Event Viewer. If there are several error events, locate the one that happened first in time, it is likely to be the one with the more precise description of the failure. Other error events are usually consequences of the first one. The following are some of the more common error events and a description of the action you can take to remedy the error:

- Service Specific Error 0x1001 (4097) - "Error in DataKinetics GCT system"

  Check that GCTLOAD was running before you started the Intel® Dialogic system service.
- Service Specific Error 0x2003 (8195) - "No Application Module Ids defined"

  Check that you have added the SS7 service(s). In Windows systems, SS7 services are added using the Intel® Dialogic Configuration Manager (DCM).

Also, check that you have specified a valid list of comma delimited module IDs to be used by Global Call SS7 applications.

- Service Specific Error 0x2009 (8201) - "Group claims a time slot that is already in use"

  Check in your *config.txt* file that no two circuit groups are defined on the same physical Intel® NetStructure™ SS7 board trunk (dkB1 or dkB2) or, if there are, that they have non-conflicting CICMask settings (a single time slot cannot be used for both groups.)

Table 7 lists all the errors that can be returned by the SS7 server.

Table 7. SS7 Server Specific Errors

Error Code	Description
System Errors	
0x1000	No memory available
0x1001	Error in DataKinetics GCT system
0x1002	Timer error
0x1003	Error in (D)COM operation
Environment and Configuration Errors	
0x2000	Error updating ISUP registry key
0x2001	SCbus time slot request program not found
0x2002	Error executing SCbus time slot request program
0x2003	No Application Module IDs defined
0x2004	Error while creating HostSession object
0x2005	Error while creating or registering HostCallback object
0x2006	DIALOGICDIR env variable not defined
0x2007	Could not read config.txt file
0x2008	Could not recreate SS7 registry key
0x2009	Group claims a time slot that is already in use

Table 7. SS7 Server Specific Errors (Continued)

Error Code	Description
0x2100	SIU Management failed to initialize
0x2101	SIU Management failed to start RSI link(s)
0x2203	Error reading Third-Party Tool registry
0x2204	Error decoding Third-Party Tool registry
0x2205	Reserved time slot region was too small
Intel® Dialogic SS7 Configuration Errors	
0x3001	Invalid SS7 devices combination
0x3002	Data is missing from the configuration
0x3003	NCM API Failure
Configuration Errors (config.txt)	
0x4001	Circuit group name must be specified before ISUP_CFG_CCTGRP
0x4002	Invalid syntax in circuit group name
0x4003	Invalid syntax in circuit group parameter
0x4004	Invalid syntax in link parameters
0x4005	Conflicting data in MTP_LINK definition
0x4006	Invalid Linkset ID specified in MTP_LINK
0x4007	MTP_CONFIG missing before ISUP_CFG_CCTGRP
0x4008	Multiple MTP_CONFIG entries
0x4009	Invalid hex number format
0x400A	Invalid decimal number format
0x400B	No configuration file specified

6.2.2. Intel® Dialogic SS7 Server Consumes 100% of the CPU Cycles

Check that the module ID for the SS7 server (specified in DCM) is correctly defined as a LOCAL module ID in the *system.txt* file.

6.3. Debugging Tools

The following debugging tools are available:

- SS7 Call Control Library Trace
- Intel® Dialogic SS7 Server Log

6.3.1. SS7 Call Control Library Trace

The trace generated by the Global Call SS7 Call Control Library includes the following information:

- Messages received from the SS7 stack
- Messages sent to the SS7 stack
- Events sent to the application
- Call control requests from the application
- Call state changes
- Error conditions

Where applicable the concerned circuit and call are contained in the logged data. Trace entries contain time stamps in milliseconds.

The trace file is in a binary format, as opposed to plain readable text, in order to minimize the file size. A separate tool converts the binary trace to a more readable text version. If you need to read the contents of the trace file, contact customer support via the web site at *http://developer.intel.com/design/telecom/support/*.

6.3.2. Intel® Dialogic SS7 Server Log

The Intel® Dialogic SS7 server writes logging information to the following file:

%SystemRoot%\System32\DlgcS7.log

This is a text file that contains status messages received from the SS7 stack, SIU failure indications, and circuit groups activation information.

By default, logging is enabled, but it can be disabled. In Windows systems, disable logging using the Intel® Dialogic Configuration Manager.

7. Data Structure Reference

7.1. S7_MAKECALL_BLK Union

The S7_MAKECALL_BLK union contains SS7-specific parameter settings.

The S7_MAKECALLBLK_BLK union is defined as follows:

```
typedef union {
  struct ss7 {

    unsigned char trans_medium_req;
    /*
       TMR_SPEECH
       TMR_64K_UNREST
       TMR_3DOT1K_AUDIO
       TMR_64K_PREFERRED
       TMR_2_64K_UNREST
       TMR_386K_UNREST
       TMR_1536K_UNREST
       TMR_1920K_UNREST
       TMR_3_64K_UNREST
       TMR_4_64K_UNREST
       TMR_5_64K_UNREST
       TMR_7_64K_UNREST
       TMR_8_64K_UNREST
       TMR_9_64K_UNREST
       ...
       TMR_23_64K_UNREST
       TMR_25_64K_UNREST
       ...
       TMR_29_64K_UNREST
     */

    unsigned char destination_number_type;
    /*
       SS7_UNKNOWN_NUMB_TYPE    - spare
       SS7_SUBSCRIBER_NUMBER    - Subscriber number (national use)
       SS7_UNKNOWN_NATIONAL     - Unknown (national use)
       SS7_NATIONAL_NUMBER      - National (significant) number
       SS7_INTERNATIONAL_NUMBER - International number
       SS7_NETWORK_SPECIFIC     - Network-specific number (national use)
     */
```

```
unsigned char destination_number_plan;
/*
   SS7_UNKNOWN_NUMB_PLAN    - Unknown plan
   SS7_ISDN_NUMB_PLAN       - ISDN numb. plan E.164
   SS7_DATA_NUMB_PLAN       - Data numb. plan X.121
   SS7_TELEX_NUMB_PLAN      - Telex numb. plan F.69
 */

unsigned char internal_network_number;
/*
   INN_ALLOWED         - routing to internal network allowed
   INN_NOT_ALLOWED     - routing to internal network not allowed
 */

unsigned char origination_number_type;
/*
   SS7_UNKNOWN_NUMB_TYPE    - spare
   SS7_SUBSCRIBER_NUMBER    - Subscriber number (national use)
   SS7_UNKNOWN_NATIONAL     - Unknown (national use)
   SS7_NATIONAL_NUMBER      - National (significant) number
   SS7_INTERNATIONAL_NUMBER - International number
   SS7_NETWORK_SPECIFIC     - Network-specific number (national use)
 */

unsigned char origination_number_plan;
/*
   SS7_UNKNOWN_NUMB_PLAN    - Unknown plan
   SS7_ISDN_NUMB_PLAN       - ISDN numb. plan E.164
   SS7_DATA_NUMB_PLAN       - Data numb. plan X.121
   SS7_TELEX_NUMB_PLAN      - Telex numb. plan F.69
 */

char origination_phone_number[MAXPHONENUM];

unsigned char origination_present_restrict;
 /*
   PRESENTATION_ALLOWED
   PRESENTATION_RESTRICTED
   PRESENTATION_NOT_AVAILABLE
 */

unsigned char origination_screening;
 /*
   SCREEN_USER_PROVIDED
   SCREEN_USER_PROVIDED_VERIFIED
   SCREEN_USER_PROVIDED_FAILED
   SCREEN_NETWORK_PROVIDED
 */
```

```
     unsigned short calling_party_category;
    /*
       SS7_UNKNOWN_CATEGORY
       SS7_FR_OPERATOR_CATEGORY
       SS7_EN_OPERATOR_CATEGORY
       SS7_GE_OPERATOR_CATEGORY
       SS7_RU_OPERATOR_CATEGORY
       SS7_SP_OPERATOR_CATEGORY
       SS7_RESERVED_CATEGORY
       SS7_ORDINARY_SUBS_CATEGORY
       SS7_PRIORITY_SUBS_CATEGORY
       SS7_DATA_CATEGORY
       SS7_TEST_CATEGORY
       SS7_PAYPHONE_CATEGORY
     */

     unsigned short forward_call_indicators;

     /* bitmask - see defines below */
     void *usrinfo_bufp;   /* RFU */

     unsigned char satellite_indicator;
    /*
      SI_NOSATELLITES
      SI_1SATELLITE
      SI_2SATELLITES
    */

     unsigned char echo_device_indicator;
    /*
      EDI_ECHOCANCEL_NOTINCLUDED
      EDI_ECHOCANCEL_INCLUDED
    */

     unsigned char continuity_check_indicator;
    /*
      CCI_CC_NOTREQUIRED
      CCI_CC_REQUIRED
      CCI_CC_ONPREVIOUS
    */


long rfu[6];          /* RFU */

} ss7;
} S7_MAKECALL_BLK,  *S7_MAKECALL_BLK_PTR;
```

NOTE: The comment /* bitmask - see defines below */ in the preceding code listing refers to the fact that the bitmask is created using an OR operation on the defines from the header file.

Table 8. S7_MAKECALL_BLK Parameters

Field	Description
trans_medium_req	Specifies the format of the transmission medium requirement. Possible values are: • TMR_SPEECH - speech • TMR_64K_UNREST - 64 kbps unrestricted • TMR_3DOT1K_AUDIO - 3.1 KhZ audio • TMR_64K_PREFERRED - 64 kbps preferrred • TMR_2_64K_UNREST - 2x 64 kbps unrestricted • TMR_386K_UNREST - 386 kbps unrestricted • TMR_1536K_UNREST - 1536 kbps unrestricted • TMR_1920K_UNREST - 1920 kbps unrestricted • TMR_3_64K_UNREST - 3x 64 kbps unrestricted • TMR_4_64K_UNREST - 4x 64 kbps unrestricted • TMR_5_64K_UNREST - 5x 64 kbps unrestricted • TMR_7_64K_UNREST - 7x 64 kbps unrestricted • TMR_8_64K_UNREST - 8x 64 kbps unrestricted • TMR_9_64K_UNREST - 9x 64 kbps unrestricted … • TMR_23_64K_UNREST - 23x 64 kbps unrestricted • TMR_25_64K_UNREST- 9x 64 kbps unrestricted … • TMR_29_64K_UNREST- 9x 64 kbps unrestricted
destination_number_type	Specifies the destination number type. Possible values are: • SS7_UNKNOWN_NUMB_TYPE - spare • SS7_SUBSCRIBER_NUMBER - Subscriber number (national use) • SS7_UNKNOWN_NATIONAL - Unknown (national use) • SS7_NATIONAL_NUMBER - National (significant) number • SS7_INTERNATIONAL_NUMBER - International number • SS7_NETWORK_SPECIFIC - Network-specific number (national use)

Table 8. S7_MAKECALL_BLK Parameters (Continued)

Field	Description
destination_number_plan	Specifies the destination number plan. Possible values are: • SS7_UNKNOWN_NUMB_PLAN - Unknown plan • SS7_ISDN_NUMB_PLAN - ISDN number plan E.164 • SS7_DATA_NUMB_PLAN - Data number plan X.121 • SS7_TELEX_NUMB_PLAN - Telex number plan F.69
internal_network_number	Specifies whether routing is allowed to an internal network. Possible values are: • INN_ALLOWED - routing to internal network allowed • INN_NOT_ALLOWED - routing to internal network not allowed
origination_number_type	Specifies the origination number type. Possible values are: • SS7_UNKNOWN_NUMB_TYPE - spare • SS7_SUBSCRIBER_NUMBER - Subscriber number (national use) • SS7_UNKNOWN_NATIONAL - Unknown (national use) • SS7_NATIONAL_NUMBER - National (significant) number • SS7_INTERNATIONAL_NUMBER - International number • SS7_NETWORK_SPECIFIC - Network-specific number (national use)
origination_number_plan	Specifies the origination number plan. Possible values are: • SS7_UNKNOWN_NUMB_PLAN - Unknown plan • SS7_ISDN_NUMB_PLAN - ISDN number plan E.164 • SS7_DATA_NUMB_PLAN - Data number plan X.121 • SS7_TELEX_NUMB_PLAN - Telex number plan F.69
origination_phone_number [MAXPHONENUM]	Specifies the calling party address. If not specified, default to the address set using **gc_SetCallingNum( )** or **gc_SetParm( )**.
origination_present_restrict	Specifies the calling party address presentation restrictions. Possible values are: • PRESENTATION_ALLOWED - Presentation allowed. • PRESENTATION_RESTRICTED - Presentation restricted. • PRESENTATION_NOT_AVAILABLE - Address not available.

Table 8. S7_MAKECALL_BLK Parameters (Continued)

Field	Description
origination_screening	Specifies calling party address screening. Possible values are: • SCREEN_USER_PROVIDED - Address is user provided, not verified (National use only). • SCREEN_USER_PROVIDED_VERIFIED - Address is user provided, verified and passed. • SCREEN_USER_PROVIDED_FAILED - Address is user provided, verified and failed (Notional use only). • SCREEN_NETWORK_PROVIDED - Address is network provided.
calling_party_category	Information sent in the forward direction indicating the category of the calling party and, in case of semi-automatic calls, the service language to be spoken by the incoming, delay and assistance operators. Possible values are: • SS7_UNKNOWN_CATEGORY - unknown category • SS7_FR_OPERATOR_CATEGORY - French language operator • SS7_EN_OPERATOR_CATEGORY - English language operator • SS7_GE_OPERATOR_CATEGORY - German language operator • SS7_RU_OPERATOR_CATEGORY - Russian language operator • SS7_SP_OPERATOR_CATEGORY - Spanish language operator • SS7_RESERVED_CATEGORY - Reserved • SS7_ORDINARY_SUBS_CATEGORY - Ordinary subscriber • SS7_PRIORITY_SUBS_CATEGORY - Priority subscriber • SS7_DATA_CATEGORY - specifies a data call using voice-band data. • SS7_TEST_CATEGORY - Specifies a test call. • SS7_PAYPHONE_CATEGORY - Specifies a pay phone call.
forward_call_indicators	Specifies forward call indicators. Bitmask built by "ORing" defines from the header file.

Table 8. S7_MAKECALL_BLK Parameters (Continued)

Field	Description
satellite_indicator	Specifies the presence of satellites along the voice path. Possible values are: • SI_NOSATELLITES • SI_1SATELLITE • SI_2SATELLITES
echo_device_indicator	Specifies whether echo cancellation devices are being used or not. Possible values are: • EDI_ECHOCANCEL_NOTINCLUDED • EDI_ECHOCANCEL_INCLUDED
continuity_check_indicator	Specifies whether a continuity check should be performed on the circuit as part of the call, if it is being performed on a previous circuit, or if it is not requested at all. Possible values are: • CCI_CC_NOTREQUIRED • CCI_CC_REQUIRED • CCI_CC_ONPREVIOUS

7.2. S7_IE Structure

The S7_IE data structure describes an ISUP message parameter. This structure should not be used to allocate storage space for message parameters because its value field is defined as a single byte whereas an actual parameter value may be multi-byte. The S7_IE structure can be used to allocate storage for a block of parameters, if required.

The S7_IE structure is defined as follows:

```
typedef struct {
    unsigned char parm;      /* Parameter type */
    unsigned char length;    /* Number of bytes in the value part */
    unsigned char value;     /* First byte of the value part (there may be more)
*/
} S7_IE;
```

Table 9. S7_IE Structure Fields

Field	Description
parm	Parameter type
length	Number of bytes in the value part
value	First byte of the value part

7.3. S7_IE_BLK Structure

The S7_IE_BLK data structure contains ISUP message parameters.

The S7_IE_BLK structure is defined as follows:

```
typedef struct {
  short  length;                 /* must be less than MAXLEN_IEDATA  */
  char  data[S7_MAXLEN_IEDATA];  /* First IE (there may be more)     */
} S7_IE_BLK, *S7_IE_BLK_PTR;
```

S7_MAXLEN_IEDATA is a constant defined in the header file. Parameter blocks should not be longer than this value.

Table 10. S7_IE_BLK Structure Fields

Field	Description
length	IE data block length, which must be less than S7_MAXLEN_IEDATA
data[s7_MAXLEN_IEDATA]	Message parameters themselves, one after the other.

7.4. S7_SIGINFO_BLK Structure

The S7_SIGINFO_BLK data structure contains ISUP messages as returned by the **gc_GetSigInfo(**) function.

The S7_ SIGINFO_BLK structure is defined as follows:

```
typedef struct {
  short  length;           /* length of SigInfo block */
  unsigned char prim;      /* ISUP primitive */
  S7_IE  data;             /* First IE of the message (there may be more) */
} S7_SIGINFO_BLK, *S7_SIGINFO_BLK_PTR;
```

Table 11. S7_SIGINFO_BLK Structure Fields

Field	Description
length	Block length, including the "primitive" byte (prim) and the parameters (data).
prim	ISUP primitive (IAM, ANM, REL…)
data	Message parameters, one after the other.

7.5. S7PARM_CCTGRP_STATE_DATA Structure

The S7PARM_CCTGRP_STATE_DATA structure is used with the **gc_SetConfigData( )** function to perform maintenance operations on telephony circuit groups.

The S7PARM_CCTGRP_STATE_DATA structure is defined as follows:

```
typedef struct _s7parm_cctgrp_state_data {
    int            group_id;
    unsigned int   cic_mask;
    int            state;
} S7PARM_CCTGRP_STATE_DATA;
```

Table 12. S7PARM_CCTGRP_STATE_DATA Structure Fields

Field	Description
group_id	Circuit group ID that must be set to one of the group IDs specified in the *config.txt* file.
cic_mask	Specifies the circuits in the group to which an operation applies. Each circuit in the group is represented by a bit in this field.
state	Specifies the maintenance operation to perform. See Section 5.11, "Circuit Group Blocking, Unblocking and Resetting", on page 77 for more information.

Appendix A:
SS7-Specific Error Codes

When a function fails, the **gc_ErrorInfo( )** function, or the **gc_ErrorValue( )** function (deprecated), can be used to retrieve error code information as follows:

- When the **gc_ErrorInfo( )** function is used, the **a_Infop** parameter is a pointer to a GC_INFO structure that contains both the standard Global Call error or result value (gcValue field), and an SS7-specific error or result value (ccValue field).
- When the **gc_ErrorValue( )** function is used, function parameters point to a standard Global Call error or result value (**gc_errorp function** parameter), and an SS7-specific error code (**cclib_errorp** function parameter).

Table 13 lists SS7-specific error codes and describes the meaning of each code. These error codes are defined in the [Dialogic installation directory]\inc\cc_s7.h header file.

Table 13. SS7-Specific Error Codes

Error Code	Value	Description
S7ERR_NO_SESSION	0x8001	No session was established with SS7 server
S7ERR_UNSUPPORTED	0x8002	Function not supported
S7ERR_INV_PARM	0x8003	Invalid parameter
S7ERR_INV_INFO_ID	0x8004	Invalid Call Info ID
S7ERR_INV_PARM_ID	0x8005	Invalid Parameter ID (in Set/GetParm)
S7ERR_INV_SIGINFO_ SIZE	0x8006	Invalid SigInfo Buffer size
S7ERR_LDEV_RELATED	0x8007	Event is related to a LineDevice (=> no CRN, no SigInfo)
S7ERR_NO_SIGINFO	0x8008	No SigInfo was associated with the event

Table 13. SS7-Specific Error Codes (Continued)

Error Code	Value	Description
S7ERR_NO_ SCBUSCONNECTOR	0x8009	Device does not support routing functions
S7ERR_INV_DEVNAME	0x800A	Invalid Device Name
S7ERR_INV_STATE	0x800B	Invalid State (Call/LineDev)
S7ERR_INV_CRN	0x800C	Invalid CRN
S7ERR_INV_CID	0x800D	Internal Error
S7ERR_INV_LINEDEV	0x800E	Invalid LineDevice
S7ERR_INV_TRUNKDEV	0x800F	Invalid TrunkDevice
S7ERR_INV_CHANNEL	0x8010	TrunkDevice has no such channel (ts)
S7ERR_NO_BASE_TS	0x8011	BaseTimeSlot not defined for the Trunk
S7ERR_TLS_NULL	0x8012	ThreadLocalStorage is NULL
S7ERR_PING_EVENT	0x8013	System Error
S7ERR_MSGQ_FULL	0x8014	Internal Error
S7ERR_INV_PARM_SIZE	0x8015	Internal Error
S7ERR_SRL	0x8016	SRL Error
S7ERR_SRL_PUTEVT	0x8017	SRL PutEvt Error
S7ERR_DTI_GENERIC	0x8018	Unspecified DTI error
S7ERR_DTI_OPEN	0x8019	Error opening DTI device
S7ERR_DTI_GETXMIT	0x801A	Error getting DTI TX time slot
S7ERR_DTI_LISTEN	0x801B	Error listening on DTI device
S7ERR_DTI_UNLISTEN	0x801C	Error unlistening on DTI device
S7ERR_LOG_ATTACH	0x801D	Error attaching file to logger
S7ERR_NOMEM	0x801E	Out of memory

Table 13. SS7-Specific Error Codes (Continued)

Error Code	Value	Description
S7ERR_GCT_SYSTEM	0x801F	Error in DataKinetics GCT System
S7ERR_COM_SYSTEM	0x8020	Error in COM system
S7ERR_TIMER_INIT	0x8021	Error initializing Timer sub-system
S7ERR_TIMER_ACTIVE	0x8022	Attempt to start an already active timer
S7ERR_NO_MORE_CRN	0x8023	Too many CRNs allocated on the LineDevice
S7ERR_ISUP_CODING	0x8024	Generic error while coding ISUP message
S7ERR_ISUP_DECODING	0x8025	Generic error while decoding ISUP message
S7ERR_INV_MODE	0x8026	SYNC/ASYNC Mode not supported
S7ERR_OPEN_VOICE	0x8027	Error opening voice device (in gc_OpenEx)
S7ERR_NO_VOICE	0x8028	No voice resource attached
S7ERR_VOX_LISTEN	0x8029	Error in routing voice resource (dx_listen function failed)
S7ERR_VOX_GETXMIT	0x802A	Error in routing voice resource (dx_getxmitslot function failed)
S7ERR_INIT_EVTMSK	0x802B	Internal error
S7ERR_CIRCUIT_IN_USE	0x802C	Circuit is already in use in another process
S7ERR_SERVICE_NOT_READY	0x802D	SS7 server is not running or not correctly initialized
S7ERR_NOT_ATTACHED	0x802E	Internal error

Table 13. SS7-Specific Error Codes (Continued)

Error Code	Value	Description
S7ERR_WATCHDOG_FAIL	0x802F	Internal error
S7ERR_NO_MORE_DIGITS	0x8030	No additional digit can be obtained
S7ERR_GC_CME	0x8031	Internal error
S7ERR_GC_DB	0x8032	Internal error
S7ERR_SRL_DEPOSIT	0x8033	Internal error
S7ERR_UNKNOWN	0x80FF	Unknown Error

Appendix B:
Call Setup and Call Release Scenarios

Each scenario is presented in tabular format. The tables provide the following information:

- **Application** - Lists the functions called by the application
- **Libgcs7** - Describes SS7 call control library activities including messages sent to the SS7 stack and events sent to application
- **State** - Lists current state
- **Stack** - Describes messages sent by the SS7 stack to call control library and to network interface
- **Network** - Lists the ISUP messages exchanged between the network and the device

NOTE: All scenarios described in this appendix operate in asynchronous mode.

Network Initiated Inbound Call

Incoming call notification is received as an event. The **gc_WaitCall( )** function should be issued once.

Application	Libgcs7	State	Stack	Network
gc_WaitCall( ) -->	Prepare for handling IAM messages	NULL		
	CRN assigned GCEV_OFFERED <--	OFFERED	IAM <--	IAM <--
gc_GetDNIS( ) (optional) -->				
	Return immediately with DNIS <--			

Application	Libgcs7	State	Stack	Network
gc_GetANI( ) (optional) -->				
	Return immediately with ANI <--			
gc_AcceptCall( ) (optional) -->	ACM -->		ACM -->	ACM -->
	GCEV_ACCEPT <--	ACCEPTED		
gc_AnswerCall( ) -->	ANM * -->		ANM * -->	ANM * -->
	GCEV_ANSWERED <--	CONNECTED		
NOTE: * = Or CON in case no gc_AcceptCall() has been made and circuit is in ITU mode				

Network Terminated Call

Application	Libgcs7	State	Stack	Network
		CONNECTED		
	GCEV_DISCONNECTED <--	DISCONNECTED	REL <--	REL <--
	REL * -->			
gc_DropCall( ) -->	RLC -->	IDLE	RLC -->	RLC -->
	GCEV_DROPCALL <--		RLC * <--	
gc_ReleaseCallEx( ) -->	Release CRN and call resources	NULL		
NOTE: * = The "new" release procedure of the DataKinetics software is used				

Application Initiated Outbound Call

Application	Libgcs7	State	Stack	Network
		NULL		
gc_MakeCall( ) -->	CRN assigned IAM -->	DIALING	IAM -->	IAM -->
	GCEV_ALERTING <--	ALERTING	ACM <--	ACM <--
	GCEV_CONNECTED <--	CONNECTED	ANM <--	ANM <--

Or, if the ACM contains “no indication”:

Application	Libgcs7	State	Stack	Network
		NULL		
gc_MakeCall( ) -->	CRN assigned IAM -->	DIALING	IAM -->	IAM -->
			ACM <--	ACM <--
	GCEV_ALERTING <--	ALERTING	CPG Alerting event <--	CPG Alerting <--
	GCEV_CONNECTED <--	CONNECTED	ANM <--	ANM <--

This following scenario is also possible in ITU-T operation:

Application	Libgcs7	State	Stack	Network
		NULL		
gc_MakeCall( ) -->	CRN assigned IAM -->	DIALING	IAM -->	IAM -->
	GCEV_CONNECTED <--	CONNECTED	CON <--	CON <--

Application Terminated Call

Application	Libgcs7	State	Stack	Network
		CONNECTED		
gc_DropCall( ) -->	REL -->		REL -->	REL -->
	GCEV_DROPCALL <--	IDLE	RLC <--	RLC <--
gc_ReleaseCallEx( ) -->	Release CRN and call resources	NULL		

Network Rejects Outgoing Call

Application	Libgcs7	State	Stack	Network
gc_MakeCall( ) -->	CRN assigned IAM -->	DIALING	 IAM -->	 IAM -->
	GCEV_DISCONNECTED <--	DISCONNECTED	REL <--	REL <--
	REL * -->			
gc_DropCall( ) -->	RLC -->		RLC -->	RLC -->
	GCEV_DROPCALL <--	IDLE		
gc_ReleaseCallEx( ) -->	Release CRN and call resources	NULL		
NOTE: * = The "new" release procedure of the DataKinetics software is used				

Application Rejects Incoming Call

Application	Libgcs7	State	Stack	Network
		NULL		
	CRN assigned GCEV_OFFERED <--	OFFERED	IAM <--	IAM <--
gc_GetDNIS( ) (optional)<br-->				
	Return immediately with DNIS <--			
gc_DropCall( )<br-->	REL<br-->	REL<br-->	REL<br-->	REL<br-->
	GCEV_DROPCALL <--	IDLE	RLC <--	RLC <--
gc_ReleaseCallEx( )<br-->	Release CRN and call resources			

Glare (Call Collision)

The glare condition occurs when both an incoming and outgoing call requests the same time slot. The following scenario illustrates a glare condition where priority is given to the incoming call.

Application	Libgcs7	State	Stack	Network
gc_MakeCall( )<br-->	CRN # 1 assigned IAM<br-->	NULL	IAM<br-->	IAM<br-->
	GCEV_DISCONNECTED for CRN # 1 <--	1 DISCONNECTED	IAM <--	IAM <--
NOTE: * = gc_ReleaseCallEx() is done upon reception of GCEV_DROPCALL				

Application	Libgcs7	State	Stack	Network
	CRN # 2 assigned GCEV_OFFERED 2 <--	2 OFFERED		
gc_DropCall(1) -->				
	GCEV_DROPCALL 1 <--	1 IDLE		
gc_AcceptCall(2) (optional) -->	ACM -->		ACM -->	ACM -->
	GCEV_ACCEPT 2 <--	2 ACCEPTED		
gc_ReleaseCallEx(1)*	CRN #1 released	1 NULL		
gc_AnswerCallEx(2) -->	ANM -->		ANM -->	ANM -->
	GCEV_ANSWERED 2 <--	2 CONNECTED		
NOTE: * = gc_ReleaseCallEx() is done upon reception of GCEV_DROPCALL				

As indicated in Section 5.2, "Handling of Glare Conditions", on page 65, the application must be ready to receive events for two different CRNs. However, for the CRN corresponding to the failed outgoing call, the application only has to respond to the GCEV_DROPCALL event by calling **gc_ReleaseCallEx( )**. It does not really have to handle two calls. Indeed it is first notified that call is disconnected.

Appendix C:
Sample Configuration Files

This appendix provides sample *system.txt* and *config.txt* files for the following configurations:

- PCCS6 board
- SIU with connection to one host
- Dual-Resilient SIU with connection to one host
- Dual-Resilient SIU with connection to two hosts

NOTE: The term *Septel* in configuration files relates to Intel® NetStructure™ SS7 boards.

See the *DSC131/DSC231 User Manual* for explanations of the meaning of each command and associated parameters in the *config.txt* file.

PCCS6 System File

```
* Sample system.txt for Dialogic GC/SS7 on PCCS6 system
*
* Modules running on the host:
*
LOCAL          0x00      * Timer Task
LOCAL          0x20      * ssd - Board Interface task
LOCAL          0x21      * ssd_poll
LOCAL          0x3d      * mtpsl
LOCAL          0x4d      * Dialogic SS7 Service
LOCAL          0x5d      * GC/SS7 Application 1
LOCAL          0x6d      * GC/SS7 Application 2
LOCAL          0xcf      * s7_mgt
LOCAL          0xef      * s7_log
*
* Modules running on the board (all redirected via ssd):
*
REDIRECT       0x10    0x20  * PCM/SCbus/Clocking control module
REDIRECT       0x71    0x20  * MTP2 module
REDIRECT       0x22    0x20  * MTP3 module
REDIRECT       0x23    0x20  * ISUP module.
REDIRECT       0x4a    0x20  * TUP/NUP module
REDIRECT       0x14    0x20  * TCAP module
REDIRECT       0x8e    0x20  * On-board management task
```

```
*
* Redirection of status:
*
REDIRECT       0xdf    0x4d  * LIU/MTP2 status messages to DlgcS7
*
* Now start-up all local tasks:
*
FORK_PROCESS    .\SSD.EXE -d
FORK_PROCESS    .\SSD_POLL.EXE
FORK_PROCESS    .\TIM_NT.EXE
FORK_PROCESS    .\TICK_NT.EXE
FORK_PROCESS    .\S7_MGT.EXE -d
FORK_PROCESS    .\S7_LOG.EXE -m0xef
```

PCCS6 Configuration File with Circuits on a PCCS6

```
*
* Sample PCCS6 Protocol configuraiton file (config.txt)
* for Dialogic GC/SS7.
*    - 1 PCCS6 in SCBus slave mode
*    - 2 circuit groups on the PCCS6 trunks.
*    - one SS7 link on each trunk
*
*
* Physical interface parameters :
* PCCS6_BOARD <port_id> <board_id> <num_pcm> <flags> <code_file>
*
PCCS6_BOARD  0  0  0  0x00C2  isup76.dc2
*
* Configure individual E1/T1 interfaces:
* 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

* MTP Parameters :
* MTP_CONFIG <local_spc> <ssf> <options>
*
MTP_CONFIG  2 0x8  0x0000
*
* Define linksets :
* MTP_LINKSET <linkset_id> <adjacent_spc> <num_links> <flags>
*
MTP_LINKSET  0  0x1  2  0x0000
*
* Define signalling links :
* MTP_LINK <link_id> <linkset_id> <link_ref> <slc> <board_id> <blink> <stream>
<timeslot> <flags>
*
MTP_LINK  0  0  0  0  0  0  0x10  0x10  0x0006
MTP_LINK  1  0  1  1  0  1  0x11  0x10  0x0006
```

```
*
* Define a route for each remote signalling point :
* MTP_ROUTE <dpc> <linkset_id> <user_part_mask>
*
MTP_ROUTE  1  0  0x0020
*
* ISUP Parameters :
* ISUP_CONFIG <local_pc> <ssf> <user_id> <options> <num_grps> <num_ccts>
*
ISUP_CONFIG  2  0x8  0x4d  0x0474  4  64
*
* Define ISUP circuit (groups) :
* [trunk_name]
* ISUP_CFG_CCTGRP <gid> <dpc> <base_cic> <base_cid> <cic_mask> <options>
*
* [dkB1]
ISUP_CFG_CCTGRP  0  1  0x01  0x01  0x7fff7fff  0x001e
* [dkB2]
ISUP_CFG_CCTGRP  1  1  0x21  0x21  0x7fff7fff  0x001e
*
*
* End of file
*
```

PCCS6 Configuration File with Circuits and Signaling on DTI Trunks

```
*    - 1 PCCS6 in SCBus slave mode
*    - 2 circuit groups on Dialogic DTI trunks (e.g. D/300SC-E1)
*    - 2 SS7 links routed over the SCBus from timeslot 16 of Dialogic DTI trunks
*       (ISDN firmware, e.g. CTR4, is required for this, with parameter 16 set to
*        2 in corresponding PRM file)
*
*
* Physical interface parameters :
* PCCS6_BOARD <port_id> <board_id> <num_pcm> <flags> <code_file>
*
PCCS6_BOARD  0  0  0  0x00C2  isup76.dc2
*
* Configure individual E1/T1 interfaces:
* 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
*
* MTP Parameters :
* MTP_CONFIG <local_spc> <ssf> <options>
*
MTP_CONFIG  2 0x8  0x0000
*
```

```
* Define linksets :
* MTP_LINKSET <linkset_id> <adjacent_spc> <num_links> <flags>
*
MTP_LINKSET  0  0x1  2  0x0000
*
* Define signalling links :
* MTP_LINK <link_id> <linkset_id> <link_ref> <slc> <board_id> <blink> <stream>
<timeslot> <flags>
*
* [link src=dtiB1T31]
MTP_LINK  0  0  0  0  0  0  0x12  0x01  0x0006
* [link src=dtiB2T31]
MTP_LINK  1  0  1  1  0  1  0x12  0x02  0x0006
*
* Define a route for each remote signalling point :
* MTP_ROUTE <dpc> <linkset_id> <user_part_mask>
*
MTP_ROUTE  1  0  0x0020
*
* ISUP Parameters :
* ISUP_CONFIG <local_pc> <ssf> <user_id> <options> <num_grps> <num_ccts>
*
ISUP_CONFIG  2  0x8  0x4d  0x0474  4  64
*
* Define ISUP circuit (groups) :
* [trunk_name]
* ISUP_CFG_CCTGRP <gid> <dpc> <base_cic> <base_cid> <cic_mask> <options>
*
* [dtiB1]
ISUP_CFG_CCTGRP  0  1  0x01  0x01  0x7fff7fff  0x001e
* [dtiB2]
ISUP_CFG_CCTGRP  1  1  0x21  0x21  0x7fff7fff  0x001e
*
*
* End of file
*
```

System File for Global Call SS7 Single-SIU System

```
*
* Two hosts can use the same system file when runing on a single SIU system
* Module Id's running locally on the host machine:
*
LOCAL           0xb0            * rsi Module Id
LOCAL           0xef            * REM_API_ID Module Id (s7_log)
LOCAL           0xfd            * rsicmd Module Id
LOCAL           0x1d            * ctu Module Id
LOCAL           0x3d            * siucmd Module Id
LOCAL           0x0d            * ttu Module Id
LOCAL           0x4d            * Dialogic SS7 Service
```

```
LOCAL           0x5d            * GC/SS7 Application 1
LOCAL           0x6d            * GC/SS7 Application 2
*
* Redirect modules running on the SIU to RSI:
*
REDIRECT        0x20    0xb0    * SSD module Id
REDIRECT        0xdf    0xb0    * SIU_MGT module Id
REDIRECT        0x22    0xb0    * MTP3 module Id
REDIRECT        0x14    0xb0    * TCAP module Id
REDIRECT        0x33    0xb0    * SCCP module Id
REDIRECT        0x32    0xb0    * RMM module Id
REDIRECT        0x23    0xb0    * ISUP module Id
REDIRECT        0x4a    0xb0    * TUP/NUP module Id
*
REDIRECT  0xcf 0x4d * To DlgcS7
REDIRECT  0xef 0x4d * To DlgcS7
*
* Now start-up the Host tasks ....
*
FORK_PROCESS    .\rsi.exe -r.\rsi_lnk.exe  -l1
FORK_PROCESS    .\s7_log.exe -m0xef
```

Sample Configuration File for a Single SIU and Two Hosts

```
*   DSC231 Protocol Configuration File (config.txt)
*   Refer to the DSC131/DSC231 User Manual.
*
*   Single SIU configuration with 2 Hosts
*   One PCCS6 in the SIU. One SS7 link to the adjacent PC.
*   Host 0 has one circuit group on Dialogic DTI boards ("dtiB1" on host 0)
*   Host 1 has one circuit group on Dialogic DTI boards ("dtiB1" on host 1)
*
*   SIU commands :
*   Set the SIU instance. Set to SIUA for standalone, SIUA or SIUB for dual
operation.
*   SIU_INSTANCE <instance_token> = SIUA | SIUB
*
SIU_INSTANCE SIUA
*
*   Define the network address of this SIU :
*   SIU_ADDR <network_address>
*
SIU_ADDR 146.152.183.246
*
*   Define the number of hosts that this SIU will connect to :
*   SIU_HOSTS <num_hosts>
*
SIU_HOSTS 2
*
*   Set physical Interface Parameters :
```

```
*   PCCS6_BOARD <port_id> <bpos> <num_pcm> <flags>
*   PCCS3_BOARD <port_id> <bpos> <num_pcm> <flags>
*
PCCS6_BOARD 0 4 0 0x0002
*
*   MTP Parameters :
*   MTP_CONFIG <local_spc> <ssf> <options>
*
MTP_CONFIG  2  0x8  0x0001
*
*   Define linksets :
*   MTP_LINKSET <linkset_id> <adjacent_spc> <num_links> <flags> <local_spc> <ssf>
*
MTP_LINKSET  0  1  1  0x0000  2  0x8
*
*   Define signalling links :
*   MTP_LINK <link_id> <linkset_id> <link_ref> <slc> <bpos> <blink> <stream>
<timeslot> <flags>
*
MTP_LINK  0  0  0  0  4  0  0x10  0x10  0x06
*
*   Define a route for each remote signalling point :
*   MTP_ROUTE <dpc> <linkset_id> <user_part_mask>
*
MTP_ROUTE  1  0  0x0020
*
*   ISUP Parameters :
*   ISUP_CONFIG <local_pc> <ssf> <user_id> <options> <num_grps> <num_ccts>
*
ISUP_CONFIG  2  0x8  0x4d  0x0474  8  96
*
*   Define ISUP circuit (groups) :
*   ISUP_CFG_CCTGRP <gid> <dpc> <base_cic> <base_cid> <cic_mask> <options>
<host_id>
*
* [dtiB1]
ISUP_CFG_CCTGRP  0  1  0x01  0x01  0x7fff7fff  0x0002 0x00
* [dtiB1]
ISUP_CFG_CCTGRP  1  1  0x21  0x21  0x7fff7fff  0x0002 0x01
*
*   Cross Connections :
*   These commands control the connection of voice channels through
*   the SIU.
*
STREAM_XCON 4 16 17 3 0xfffefffe 0x00
*
* End of file*
```

System File on a Dual-Resilient SIU System

```
*
* Sample System.txt for Dialogic GC/SS7 on a Dual SIU system
*
*
* Module Id's running locally on the host machine:
*
LOCAL           0xb0            * rsi Module Id
LOCAL           0xef            * REM_API_ID Module Id (s7_log)
LOCAL           0xfd            * rsicmd Module Id
LOCAL           0x1d            * ctu Module Id
LOCAL           0x3d            * siucmd Module Id
LOCAL           0x0d            * ttu Module Id
LOCAL           0x4d            * Dialogic SS7 Service
LOCAL           0x5d            * GC/SS7 Application 1
LOCAL           0x6d            * GC/SS7 Application 2
*
* Redirect modules running on the SIU to RSI:
*
REDIRECT        0x20    0xb0    * SSD module Id
REDIRECT        0xdf    0xb0    * SIU_MGT module Id
REDIRECT        0x22    0xb0    * MTP3 module Id
REDIRECT        0x14    0xb0    * TCAP module Id
REDIRECT        0x33    0xb0    * SCCP module Id
REDIRECT        0x32    0xb0    * RMM module Id
REDIRECT        0x23    0xb0    * ISUP module Id
REDIRECT        0x4a    0xb0    * TUP/NUP module Id
*
REDIRECT  0xcf 0x4d * To DlgcS7
*
* Now start-up the Host tasks ....
*
FORK_PROCESS    .\rsi.exe -r.\rsi_lnk.exe  -l1
FORK_PROCESS    .\s7_log.exe -m0xef
```

Configuration File for Dual-Resilient SIU and a Single Host: Configuration for SIU A

```
*
*   DSC231 Protocol Configuration File (config.txt)
*   Refer to the DSC131/DSC231 User Manual.
*
*   Dual-Resilient SIU configuration with 1 Host
*   One PCCS6 in each SIU. One link interconnects both SIUs.
*   On the host, 2 circuit groups terminate on Dialogic DTI boards
*   Under normal condition (both SIU ok), group "dtiB1" is managed by SIU A
*   and group "dtiB2" is managed by SIU B.
```

```
*   This file is for SIU A.
*
*
*   SIU commands :
*   Set the SIU instance. Set to SIUA for standalone, SIUA or SIUB for dual
operation.
*   SIU_INSTANCE <instance_token> = SIUA | SIUB
*
SIU_INSTANCE SIUA
*
*   Define the network address of this SIU :
*   SIU_ADDR <network_address>
*
SIU_ADDR 146.152.183.246
*
*   Define the network address of the partner SIU (dual operation only) :
*   SIU_REM_ADDR <remote_address>
*
SIU_REM_ADDR 146.152.183.188
*
*   Define the number of hosts that this SIU will connect to :
*   SIU_HOSTS <num_hosts>
*
SIU_HOSTS 1
*
*   Set physical Interface Parameters :
*   PCCS6_BOARD <port_id> <bpos> <num_pcm> <flags>
*   PCCS3_BOARD <port_id> <bpos> <num_pcm> <flags>
*
PCCS6_BOARD 0 4 0 0x0002
*
*   MTP Parameters :
*   MTP_CONFIG <local_spc> <ssf> <options>
*
MTP_CONFIG  2  0x8  0x0001
*
*   Define linksets :
*   MTP_LINKSET <linkset_id> <adjacent_spc> <num_links> <flags> <local_spc> <ssf>
*
MTP_LINKSET  0  1  1  0x0000  2  0x8
*   Inter-siu linkset:
MTP_LINKSET  1  2  1  0x8000  2  0x8
*
*   Define signalling links :
*   MTP_LINK <link_id> <linkset_id> <link_ref> <slc> <bpos> <blink> <stream>
<timeslot> <flags>
*
MTP_LINK  0  0  0  0  4  0  0x10  0x10  0x0006
* [link inter-siu]
MTP_LINK  1  1  0  0  4  1  0x11  0x10  0x0006
*
*   Define a route for each remote signalling point :
```

```
*   MTP_ROUTE <dpc> <linkset_id> <user_part_mask>
*
MTP_ROUTE  1  0  0x0020
*
*   ISUP Parameters :
*   ISUP_CONFIG <local_pc> <ssf> <user_id> <options> <num_grps> <num_ccts>
*
ISUP_CONFIG  2  0x8  0x4d  0x0474  8  96
*
*   Define ISUP circuit (groups) :
*   [device_name <SIUA|SIUB>]
*   ISUP_CFG_CCTGRP <gid> <dpc> <base_cic> <base_cid> <cic_mask> <options>
<host_id>
*
* [dtiB1 SIUA]
ISUP_CFG_CCTGRP  0  1  0x01  0x01  0x7fff7fff  0x0002 0x00
* [dtiB2 SIUB]
ISUP_CFG_CCTGRP  1  1  0x21  0x21  0x7fff7fff  0x0002 0x00
*
*   Cross Connections :
*   These commands control the connection of voice channels through
*   the SIU.
*
STREAM_XCON 4 16 17 3 0xfffefffe 0x00
*
*
* End of file
*
```

Configuration File for Dual-Resilient SIU and a Single Host: Configuration for SIU B

```
*   DSC231 Protocol Configuration File (config.txt)
*   Refer to the DSC131/DSC231 User Manual.
*
*   Dual-Resilient SIU configuration with 1 Host
*   One PCCS6 in each SIU. One link interconnects both SIUs.
*   On the host, 2 circuit groups terminate on Dialogic DTI boards
*   Under normal condition (both SIU ok), group "dtiB1" is managed by SIU A
*   and group "dtiB2" is managed by SIU B.
*   This file is for SIU B.
*
*
*   SIU commands :
*   Set the SIU instance. Set to SIUA for standalone, SIUA or SIUB for dual
operation.
*   SIU_INSTANCE <instance_token> = SIUA | SIUB
*
SIU_INSTANCE SIUB
```

```
*
*   Define the network address of this SIU :
*   SIU_ADDR <network_address>
*
SIU_ADDR 146.152.183.188
*
*   Define the network address of the partner SIU (dual operation only) :
*   SIU_REM_ADDR <remote_address>
*
SIU_REM_ADDR 146.152.183.246
*
*   Define the number of hosts that this SIU will connect to :
*   SIU_HOSTS <num_hosts>
*
SIU_HOSTS 1
*
*   Set physical Interface Parameters :
*   PCCS6_BOARD <port_id> <bpos> <num_pcm> <flags>
*   PCCS3_BOARD <port_id> <bpos> <num_pcm> <flags>
*
PCCS6_BOARD 0 4 0 0x0007
*
*   MTP Parameters :
*   MTP_CONFIG <local_spc> <ssf> <options>
*
MTP_CONFIG  2  0x8  0x0001
*
*   Define linksets :
*   MTP_LINKSET <linkset_id> <adjacent_spc> <num_links> <flags> <local_spc> <ssf>
*
MTP_LINKSET  0  1  1  0x0000  2  0x8
*   Inter-siu linkset:
MTP_LINKSET  1  2  1  0x8000  2  0x8
*
*   Define signalling links :
*   MTP_LINK <link_id> <linkset_id> <link_ref> <slc> <bpos> <blink> <stream>
<timeslot> <flags>
*
MTP_LINK  0  0  0  1  4  0  0x10  0x10  0x0006
* [link inter-siu]
MTP_LINK  1  1  0  0  4  1  0x11  0x10  0x0006
*
*   Define a route for each remote signalling point :
*   MTP_ROUTE <dpc> <linkset_id> <user_part_mask>
*
MTP_ROUTE  1  0  0x0020
*
*   ISUP Parameters :
*   ISUP_CONFIG <local_pc> <ssf> <user_id> <options> <num_grps> <num_ccts>
*
ISUP_CONFIG  2  0x8  0x4d  0x0474  8  96
*
```

```
*   Define ISUP circuit (groups) :
*   [device_name <SIUA|SIUB>]
*   ISUP_CFG_CCTGRP <gid> <dpc> <base_cic> <base_cid> <cic_mask> <options>
<host_id>
*
* [dtiB1 SIUA]
ISUP_CFG_CCTGRP  0  1  0x01  0x01  0x7fff7fff  0x0002 0x00
* [dtiB2 SIUB]
ISUP_CFG_CCTGRP  1  1  0x21  0x21  0x7fff7fff  0x0002 0x00
*
*   Cross Connections :
*   These commands control the connection of voice channels through
*   the SIU.
*
STREAM_XCON 4 16 17 3 0xfffefffe 0x00
*
*
* End of file
*
```

Glossary

CT Bus: A time division multiplex (TDM) bus that provides 1024, 2048, or 4096 time slots for exchanging voice, fax, or other network resources on a PCI (H.100) or CompactPCI (H.110) backplane. The Enterprise Computer Telephony Forum (ECTF) developed the H.100 hardware compatibility specification that defined the CT Bus, a high-performance mezzanine bus. The CT Bus works with both SCbus and Multivendor Integration Protocol (MVIP) compatible products. The ECTF implementation of the CT Bus for CompactPCI bus is called the H.110 standard.

DataKinetics Environment: A collective name for the DataKinetics system environment and the SS7 stack.

DCM: Intel® Dialogic Configuration Manager. A Windows application the enables the configuration of Intel® Dialogic products.

Global Call SS7 Software: The software and libraries that implement Global Call on SS7.

DPC: Destination Point Code. Identifies the address (point code) of the SS7 network node to which a Message Signal Unit (MSU) should be directed.

DTI: A generic term for an Intel® Dialogic network interface card, such as, D/300SC-E1, D/240SC-T1, DTI/300SC-E1, etc.

E-1: A digital transmission link that carries information at the rate of 2,048 Mbps. This is the rate used by European carriers to transmit thirty 64 Kbps digital channels for voice or data calls, plus one 64 Kbps channel for signaling, and one 64 Kbps channel for framing (synchronization) and maintenance.

IPC: Inter Process Communication. In a DataKinetics environment, IPC refers to the method by which modules communicate with each other using messages.

ISUP: ISDN User Part. A layer in the SS7 protocol stack. Defines the messages and protocol used in the establishment and tear down of voice and data calls over the public switched network, and to manage the trunk network on which they rely.

ISDN: Integrated Services Digital Network. A service that offers simultaneous digital data and voice communication over a single copper pair wire in residential and business phone connections. There are two basic flavors, BRI (Basic Rate Interface) which is 144 Kbps and designed for the desktop, and PRI (Primary Rate Interface) which is 1.544 Mbps and designed for telephone switches, computer telephony and voice processing systems.

Message Transfer Part: Layers 1 to 3 of the SS7 protocol stack equivalent to the Physical, Data Link and Network layers in the OSI protocol stack. See also MTP 1, MTP 2 and MTP 3.

MTP1: Message Transfer Part Level 1. Defines the physical and electrical characteristics of the signaling links of the SS7 network. Signaling links use DS0 channels and carry raw signaling data at a rate of 56 Kbps or 64 Kbps (56 Kbps is currently the more common implementation).

MTP2: Message Transfer Part Level 2. Provides link-layer functionality. Ensures that two end points of a signaling link can reliably exchange signaling messages. It provides error checking, flow control and sequence checking.

MTP3: Message Transfer Part Level 3. Provides network-layer functionality. Ensures that messages can be delivered between signaling points across the SS7 network regardless of whether the signaling points are directly connected. It provides node addressing, routing, alternate routing and congestion control.

OPC: Originating Point Code. Identifies the address (point code) of the SS7 network node from which a Message Signal Unit (MSU) originated.

PSTN: Public Switched Telephony Network. The worldwide voice telephone network accessible to all those with telephones and access privileges.

PCCS6: An Intel® NetStructure™ SS7 ISA board solution.

SCbus: The standard bus for communicating within an SCSA node. The SCbus features a hybrid bus architecture consisting of a serial message bus for control and signaling, and a 16-wire TDM data bus.

SCCP: Signal Connection Control Part. A layer in the SS7 protocol stack that allows a software application at a specific node in an SS7 network to be addressed.

It also supports Global Title Translation which frees an originating signaling point from having to know every possible destination to which a message may have to be routed.

SCP: Service Control Point. Databases that provide information necessary for advanced call-processing capabilities.

Signaling Link: A signaling data link is a bidirectional transmission path for signaling, comprising two data channels operating together in opposite directions at the same data rate.

SIU: The Intel® NetStructure™ SS7 server solution.

SP: Signaling Point. Any point in a signaling network capable of handling SS7 control messages. Examples of Signaling Points are: SSP (Signal Switching Point), STP (Signal Transfer Point), and SCP (Signal Control Point).

SS7: Signaling System Number 7. A common channel signaling standard that defines the procedures and protocols required for the connection of network elements in the Public Switched Telephone Network (PSTN).

SSP: Signal Switching Point. Telephone switches (end offices or tandems) equipped with SS7-capable software and terminating signaling links. They generally originate, terminate or switch calls.

STP: Signal Transfer Point. A signaling point capable of routing control messages to another signaling point. STPs receive and route incoming signaling messages towards the proper destination and perform specialized routing functions.

RSI: Remote Socket Interface.

T-1: A digital transmission link with a capacity of 1.544 Mbps (mega bits per second). T-1 uses two pairs of normal twisted wires and can handle twenty-four voice conversations, each one digitized at 64 Kbps.

TCAP: Transaction Capabilities Part. A layer in the SS7 protocol stack that defines the messages and protocol used to communicate between applications (deployed as subsystems) in SS7 nodes. TCAP is used for database services such as

calling card, 800, and AIN, as well as switch-to-switch services including Repeat Dialing and Call Return.

TUP: Telephone User Part. The predecessor to ISUP (Integrated Services User Part). TUP was employed for call control purposes within and between national networks, both wireline and wireless. ISUP adds support for data, advanced ISDN, and IN (Intelligent Networks). See also ISUP.

User Part: A generic name given to an SS7 stack protocol at layer 4 or above, such as, ISUP, TUP, ICAP, MAP etc.

Index

G

I

M

N

O

P

R

S

T