Dialogic

Dialogic® Distributed Signaling Interface Components -

Software Environment Programmer's Manual

September 2013 U10SSS

www.dialogic.com

Copyright and Legal Notice

Publication Date: September 2013

Document Number: U10SSS

Revision History

Issue	Date	Description
15	30-Sep-13	Addition of IPv6 support for SIGTRAN. Addition of support for SS7LD, DNI2410TEPE2HMP, DNI1210TEPE2HMP DNI610TEPE2HMP and DN310TEPE2HMP boards under Linux. Number of supported SIGTRAN associations increased from 256 to 384.
14	26-Jun-13	Addition of support for SS7LD, DNI2410TEPE2HMP, DNI1210TEPE2HMP DNI610TEPE2HMP and DN310TEPE2HMP boards under Windows.
13	22-Feb-13	Updates to Diameter Java package.
12	11-Jan-13	Addition of Diameter support for Linux DPK. Additional SIGTRAN configuration documentation.
11	05-Nov-12	Addition of documantation for Dialogic® DSI Diameter Stack. Addition of IPv6 support for RSI. Changes to locations of 32 and 64 bit shared object libraries. General updates to reflect current Development Packages.
10	13-Apr-12	Added documentation of RSI, RSICMD and RSI_LNK and RSI message definitions. Includes enhancements to gctload –tn status displays and other minor changes throughout.
9	28-Jul-11	This release expands the scope of the manual to include all config.txt related configuration commands and full details on SIGTRAN configuration (previously contained in Dialogic® SS7 Protocols Programmer's Manual for SIGTRAN Host Software). Support for the SS7LD board is added.
8	09-Dec-10	Addition of Installation, Configuration and Execution sections
7	12-Feb-10	S7_log PCAP support documented. GCT_LOAD Verification command added. System resource parameter settings defined.
6	28-Nov-08	Rebranded Dialogic® DSI, added long message support, rolling logs and updated Solaris kernel tuning.
5	05-Dec-05	Additional information added including s7_log, s7_play, congestion management and gctload command line parameters. References to T_FRAME and R_FRAME removed.
4	02-Aug-03	Branding changed to Intel® NetStructure™.
3	28-May-99	Description of module instance and associated library functions added. Table of default module identifiers added. Module identifiers and message types reserved for user's applications added.
2	19-Feb-97	All commands in system.txt now commence with a key word. Definition of MSG made consistent with actual header file.
1	18-Jul-95	Initial Release

Note: The current version of this guide can be found at:
http://www.dialogic.com/support/helpweb/signaling

Contents

Tables

1 Introduction

Dialogic® Distributed Signaling Interface (DSI) Components is a range of hardware and software components for realization of SS7, SIGTRAN and Diameter signaling nodes and applications for use in a service provider environment. The range includes Dialogic® DSI Protocol Stacks, which are software implementations of standards-based signaling protocol layers. DSI Protocol Stacks are available for specific Dialogic® products and are suitable for use under standard commercially available operating systems including Linux, Solaris, and Windows[1] operating systems.

In a signaling node built from DSI Components (the "system"), each module in the system is implemented as a separate task within the chosen operating environment. A module implements either a layer within the DSI Protocol Stack, a User Part, or some other functional entity within the system. In general, a module supports multiple internal instances within a single process (for example multiple links, multiple circuits, or multiple transactions). The architecture supports multi-processor operation with modules being distributed between different processors.

For increased flexibility, the protocol implementation is abstracted from the underlying operating system. Each module makes a minimum demand on the host operating system using a common set of functions for all inter-process communication and resource allocation. This approach means that different layers of the DSI Protocol Stack can easily be run on different processors or machines as required.

This document is the base reference material applicable to all Dialogic® DSI board based and SIGTRAN based deployments. It introduces the fundamental architectural concepts of modules, messages and message queues and the mechanisms for inter process communication. It provides details of all the host-based utilities used to configure and maintain an operational system including full definitions of all configuration file commands and messages.

This manual also provides full installation and configuration details for use of the Development Packages for Linux, Solaris (SPARC and x86) and Windows Operating Systems.

1.1 Applicability

This manual is applicable to the following software:

Dialogic® DSI Development Package for Linux – Release 6.6.1 or later

Dialogic® DSI Development Package for Solaris – Release 5.4.0 or later

Dialogic® DSI Development Package for Windows – Release 6.5.0 or later

1.2 Related Documentation

Current software and documentation supporting Dialogic® DSI products is available at:

http://www.dialogic.com/support/helpweb/signaling

1.2.1 Dialogic® DSI SS7 Protocol Manuals

- *Dialogic® SS7 Protocols MTP2 Programmer's Manual*

[1] Note: Throughout this document, the term "Windows" is used to refer to the Windows Server 2008, Windows Server 2008 R2, and Windows 7 operating systems.

- *Dialogic® SS7 Protocols MTP3 Programmer's Manual*
- *Dialogic® SS7 Protocols ISUP Programmer's Manual*
- *Dialogic® SS7 Protocols SCCP Programmer's Manual*
- *Dialogic® SS7 Protocols TCAP Programmer's Manual*
- *Dialogic® SS7 Protocols TUP Programmer's Manual*
- *Dialogic® SS7 Protocols MAP Programmer's Manual*
- *Dialogic® SS7 Protocols IS41 Programmer's Manual*
- *Dialogic® DSI Protocol Stacks - Host Licensing User Guide*
- *Dialogic® DSI Protocol Stacks - SNMP User Manual*

1.2.2 Dialogic® DSI SIGTRAN Protocol Manuals

- *Dialogic® SS7 Protocols SCTP Programmer's Manual*
- *Dialogic® SS7 Protocols M3UA Programmer's Manual*
- *Dialogic® SS7 Protocols M2PA Programmer's Manual*
- *Dialogic® SS7 Protocols SUA Programmer's Manual*

1.2.3 Dialogic® DSI Diameter Stack Manuals

- *Dialogic® DSI Diameter Stack – DMR Programmer's Manual*
- *Dialogic® DSI Diameter Stack - Diameter Functional API Manual*

1.2.4 Dialogic® DSI Network Interface Boards Manuals

- *Dialogic® DSI SPCI Network Interface Boards Programmer's Manual*
- *Dialogic® DSI SS7HD Network Interface Boards Programmer's Manual*
- *Dialogic® DSI SS7MD Network Interface Boards Programmer's Manual*
- *Dialogic® DSI SS7LD Network Interface Boards Programmer's Manual*

1.2.5 Dialogic® DSI Signaling Servers Manuals

- *Dialogic® DSI Signaling Servers SS7G41 SIU Developers Manual*
- *Dialogic® DSI Signaling Servers SS7G41 Operators Manual*
- *Dialogic® DSI Signaling Servers SS7G41 SWS Developers Manual*
- *Dialogic® DSI Signaling Servers SS7G41 Hardware Manual*
- *Dialogic® DSI Signaling Servers SNMP User Manual*

2 Basic Concepts

This section introduces basic concepts and terminology that will be used throughout the remainder of the manual.

2.1 Modules

A module is an implementation of a particular layer in the Dialogic® DSI Protocol Stack (e.g., Dialogic® DSI MTP3 Layer), a particular user part (e.g., Dialogic® DSI ISUP Layer), or a collection of other functionality which fits together as a logical entity. A module may be a Dialogic® DSI Component or a User-supplied module.

Each module in the system runs as a separate task, process, or program (depending on the type of operating system). The module is identified by a **Module Identifier** and communicates with other modules in the system by sending **Messages** to a **Message Queue** belonging to the destination module. A set of **Library Functions** is used by the module to interact with the operating system.

A module handles multiple internal instances of the functional entity associated with the module (e.g., Dialogic® DSI MTP2 Layer handles multiple Signaling links, the Dialogic® DSI MTP3 Layer handles multiple link sets and multiple routes and the Dialogic® DSI ISUP Layer handles multiple circuits).

2.2 Module Identifiers

Each module has a module identifier (module_id), which is a logical number in the range 0 to 254, and is used to identify modules within the system for the purposes of inter-process communication. To send a message to another process, the sending module uses the module identifier of the destination process. To receive a message from a module's own message queue, it uses its own module identifier.

Some modules operate with a fixed module identifier, whereas others allow the module identifier to be specified at run-time.

The module identifiers of other modules with which a module will communicate is usually a run-time configuration option.

The module identifier is a logical value; it is not the same as an Operating System’s task id or process id (pid), which are usually allocated automatically when a process is created.

In addition to modules that are physically implemented, it is possible to use virtual module_ids that are redirected within the software environment to an actual module. This re-direction mechanism is used when messages need to be transferred to another board or module in the system or to a separate host. Messages for all processes that run on separate boards are redirected to a special local module that handles inter-board message passing. Other modules within the system do not need to know whether the modules with which they communicate are running locally or not.

A list of default module identifiers used by the Dialogic® Distributed Signaling Interface (DSI) software is given in 9.10.

2.3 Messages

Modules communicate by sending messages to other modules in the system.

The MSG message is a 'C' data structure containing a fixed format header field and a buffer for variable length parameter data.

Each hardware product in the Dialogic® Distributed Signaling Interface (DSI) Components range has a manual that details the messages appropriate for that particular product. In addition, each Dialogic® DSI Protocol Stack has a supporting Programmer's Manual, which describes the messages appropriate to that protocol.

A detailed description of the message structure is given in 5.1 Message Format.

2.4 Message Queues

Each module in the system has a single message queue that is used by other modules to send messages to the module. A message queue is a system buffer which stores messages (usually by reference) in first-in, first-out order.

Messages are read out of the message queue by the receiving module, which typically uses a blocking function call to wait until there is a message available before returning, it then processes the message and blocks until the next message is available. Input to the module is through its message queue.

2.5 Distributed Modules

Some systems require the functional entity implemented by a module to be distributed across several processors in a system. For example, a module may run on several separate boards in a single computer, each board interacting with a single module running on the computer. Alternatively, a user's application may be distributed across several host computers where each host interacts with a protocol module running on a single protocol server.

In both cases, there is a 'one too many' relationship between the distributed processors and the adjacent layer in the DSI Protocol Stack. There is a clear requirement for the single module to be able to determine from which of the distributed processors a message has been received and to which of the distributed processors a message should be sent to. This is achieved using the concept of a module **Instance**.

The module **Instance** is a number in the range of 0 to one less than the number of distributed processors. The module instance is used by the inter-board message passing process to determine which board to send the message to. When messages are received from other boards, the inter-board message passing process inserts the module instance of the board from which the message was received.

The module instance is not directly accessible as a field in the message; instead, a functional interface is provided to read and write instance information to the message. By default, the instance number is initialized to zero.

2.6 Library Functions

Host modules and user applications make use of the following set of 'system' library functions:

getm	Function to allocate a message (MSG).
relm	Function to release MSG back to the system.
GCT_send	Function to send a message to another module.

GCT_receive	Function to receive a message from a module's own message queue. The function does not return until a message is ready.
GCT_grab	Function to receive a message from a module's own message queue. This function returns immediately if there are no messages ready.
GCT_set_instance	Function to insert the module instance into a message.
GCT_get_instance	Function to extract the module instance from a message.
GCT_link	Function to attach to the message passing environment.
GCT_unlink	Function to detach from the message passing environment.
GCT_partition_ congestion	Function to determine current system congestion status.
confirm_msg	Function to confirm a message once it has been handled.

The syntax for each of these functions is described in the following section. Their usage is described below.

A module wishing to send a message to another module will first allocate a MSG structure using the **getm** function. At this stage, it is necessary to decide whether or not a confirmation message will be required and initialize the rsp_req field accordingly. Once all the message parameters have been entered into the MSG, the module calls **GCT_send** to send the message to the destination module. If the **GCT_send** function fails to send the message, the sending module must release the message back to the system using the relm function (although this will only happen when the system is incorrectly configured). When multiple destination processors are used, the module sending the message must call **GCT_set_instance** prior to calling **GCT_send** in order to write the destination module instance into the message.

The destination module will receive the message from its own message queue using either the **GCT_receive** or **GCT_grab** functions (depending on whether it wishes to block or not if no messages are available). It then processes the message. When multiple source processors are used, the module receiving the message should call **GCT_get_instance** after calling **GCT_receive** or **GCT_grab** in order to read the source module instance from the message.

When the receiving module has finished processing the message, it carries out one of two possible courses of action depending on whether or not a confirmation is required. If no confirmation is required, then the message is released back to the system using the **relm** function. If a confirmation is required, then a status value is written into the message header, the message type is changed (bit 14 is cleared), and the message is sent back to the original sending module using the **GCT_send** function. On receipt of the confirmation, the original sending module (after inspecting the status) releases the message back to the system using the **relm** function.

Note: **confirm_msg**() is a useful library function that can be called once the application has finished with a message. It determines whether or not a confirmation is required, modifies the message header accordingly, and finally calls either **GCT_send**() or **relm**() as appropriate.

In this way it is ensured that each message will eventually be released back to the system.

Example Code

- Allocating and Sending a Message

```
/*
 * Base DSI headers.
 */
#include "system.h"
#include "msg.h"
#include "sysgct.h"
/*
 * DSI protocol headers.
 */
#include "mtp_inc.h"

/*
 * MACROs for sending and receiving requests.
 */
#define NO_RESPONSE                    (0)
#define RESPONSE(mod_id)               (1 << ((mod_id) &  0x0f))
#define CONF(i)                        ((i) & ~REQUEST)

#define EXAMPLE_MODULE_ID              (0x1d)

int allocate_and_send_example(void)
{
  MSG      *m;
  u8       *pptr;

  /*
   * Allocate a MSG from the message pool.
   * In this example, a MTP3 Linkset Configuration Message.
   *
   * The rsp_req field is set to request a response from the
   * destination module Id.
   */
  if ((m = getm(MTP_MSG_CNF_LINKSET, (u16)(<LINKSET ID> << 8),
        (u16)RESPONSE(EXAMPLE_MODULE_ID), MTPCFLS_LENGTH)) != 0)
  {
    /*
     * getm() succeeds and returns a pointer to a MSG from the
     * global message pool.
     * This process now 'owns' the MSG and is responsible for
     * sending it (GCT_send) to another module, or releasing it
     * (relm).
     */
    m->hdr.src = EXAMPLE_MODULE_ID;
    m->hdr.dst = MTP_TASK_ID;

    /*
     * Initialise a memory pointer to the start of the MSG's
     * parameter area.
     */
    pptr = get_param(m);

    /*
     * Reset the MSG's parameter area to 0.
     */
    memset(pptr, 0, m->len);

    /*
     * Initialise the MSG's parameter values:
```

```
   */
  rpackbytes(pptr, MTPCFLS_adj_pc_OFF,
         (u32)<ADJACENT POINT_CODE>,
         MTPCFLS_adj_pc_SIZ);

  rpackbytes(pptr, MTPCFLS_num_links_OFF,
         (u32)<NUMBER OF LINKS>,
         MTPCFLS_num_links_SIZ);
  /*
   * MSG parameter initialization continues . . .
   */

  /*
   * Set the MSG instance.
   */
  GCT_set_instance(<INSTANCE ID>, &m->hdr);

  /*
   * Send the MSG
   */
  if (GCT_send(msg->hdr.dst, &msg->hdr) != 0)
  {
    /*
     * GCT_send() has failed.
     * Sending process retains ownerships of the MSG.
     * Release the MSG.
     */
     relm(&msg->hdr);
  }
}
```

- Receiving and Processing a Message

```
int receive_and_process_msg(void)
{
  MSG *m;

  if ((msg = (MSG *)GCT_receive(EXAMPLE_MODULE_ID)) != 0)
  {
    /*
     * GCT_receive() succeeds and returns a MSG.
     * The MSG is now owned by the receiving program.
     */
    switch (msg->hdr.type)
    {
      case CONF(MTP_MSG_CNF_LINKSET):
        /*
         * Process MTP3 Configure Linkset request's response.
         */
        . . .
        break;

      case . . .
    }

    /*
     * Release received MSG back to the system pool.
     */
    relm(&m->hdr);
  }
```

2.7 System Initialization

System initialization requires first that a pool of message buffers is created for subsequent inter-process communication. Secondly, a message queue is created for each module that will run and that any message re-direction for modules that are running remotely is initialized. Then the process can be started.

A program **gctload** exists to handle this initialization sequence. It reads input from a text file called system.txt, carries out all system initialization and starts up the processes. It then remains dormant until it receives a signal from the operating system to shutdown. Then, it terminates the processes that it started and releases any system resources back to the system in a controlled manner.

The basic operation of the **gctload** program, and the format of the text file it uses, is described in section 7.1. The **gctload** utility is available for a number of operating systems including Linux, Solaris (Sparc and x86) and Windows.

2.8 Attaching to the DSI environment

Normally, the **gctload** module, which establishes message queues and pre-allocates memory from the system for the messages used by all tasks is used to initiate all user modules. These structures are released on termination of gctload, hence the tasks that communicate using these mechanisms and the user tasks are also terminated.

If the user chooses to run applications independently, then it is necessary to confirm that the system is available before using any system resources The **GCT_link** library function can be used to obtain this confirmation. See section 6.1.8 for details.

The user must also ensure that the application does not attempt to access system resources after the GCT environment has been terminated. The **GCT_unlink** library function can be used to release all system resources. See section 6.1.9 for details.

If the user wishes to terminate and re-start an application, this can be achieved by calling **GCT_unlink** from the old instance of the application prior to termination and the calling **GCT_link** from the new application before using system resources.

2.9 System Congestion

When the host software is first run, a specified number of messages (as configured in the NUM_MSGS command in the system.txt file) are allocated from host system resources. These messages are available for inter-task communication.

User applications should read messages from their own input queue, extract the information from these messages, then release the original message structure back to the system. Hence, under normal operating conditions, the host application works to ensure that its queue is almost empty, and that all the messages are available.

If, however, more messages are received than the system can handle, then a backlog of messages in input queues will build up. Possible causes include input being received at a faster rate than can be processed by the output device or the system being unable to process received input in at the required rate.

The message handling environment monitors the number of allocated messages available as a percentage of the total message pool. If this percentage exceeds a configurable **congestion onset threshold**, the host is said to be in a congested state.

When the number of allocated messages available as a percentage of the total message pool returns to a configurable **congestion abatement threshold**, the host is said to leave the congested state.

On entering or leaving a congested state, the software environment generates a congestion notification message (**SYS_MSG_CONGESTION**), which is sent to a nominated **congestion-handling module**. This module is responsible for taking actions to slow down the source of system congestion.

The congestion handling parameters are set using the **CONG_MSG** command in the system.txt file.

The current system congestion status may also be obtained on demand using the **GCT_partition_congestion** function as detailed in section 6.1.10.

3 Installation

3.1 Introduction

This manual covers the installation and use of the software contained in the following distributions:

- Development Package for Linux
- Development Package for Solaris (x86 and SPARC)
- Development Package for Windows®

Each Development Package contains the device drivers, library functions, and header files for use by an application, a number of executables to create and maintain the DSI software environment, utilities and configuration files to configure the protocol software, the User Part Development (UPD) Package, the DSI Protocol Stacks, MIBs and the DSI Network Interface Boards code files. The installation of each package type is described in the following sections.

The UPD contains example source code to illustrate the techniques used for interfacing with the protocol modules and protocol-specific header files for use when building an application.

The DSI Network Interface Boards Code Files contain the operating software for the DSI Network Interface Boards. The appropriate code file must be downloaded by the host, to the board, at run-time. Code files have a file suffix which denotes which board product they are used in conjunction with.

Table 1. Dialogic® DSI Network Interface Board Family Code File Extensions

Dialogic® DSI Network Interface Board Family	Code File Extension	Software License Mechanism
SPCI	dc3	License Button
SS7HD	dc4	License Button
SS7MD	dc6	Host License
SS7LD	dc7	Host License

The code file must be licensed; two mechanisms exist to support licensing dependant on the board family in use:

- **License Button**

The board is used in conjunction with a software license button, which is purchased and installed on the board to determine the protocols that the user is authorized to run. The types of license buttons available are described in the appropriate DSI Network Interface Board Programmer's Manual. The license button is subsequently downloaded onto the board at run time.

- **Host License**

As indicated in the table above, some of the boards require a Host License; details on how to use a Host License are given in the Dialogic® Distributed Signaling Interface Components Host Licensing User Guide.

Some SS7 protocols also optionally may be run as Host Protocol Binaries subject to the purchase of appropriate licenses, which may be run above boards or above M2PA creating a software only architecture with SIGTRAN. refer to Section 4.1.4 Protocol Modules for further details.

The Development Package may be obtained by downloading it from the Dialogic website, see Section 1.1 Related Documentation, and must be copied onto the target host machine maintaining binary file integrity; possible transfer methods include copying using transferable media and ftp.

3.2 Software Installation for Linux

The Development Package for Linux is distributed electronically. The distribution is in the form of a single compressed file called dpklnx.Z.

The Development Package for Linux includes, in a single distribution, the software required by users of the Dialogic® DSI SS7 Boards and Dialogic® DSI Protocol Stacks.

The Development Package includes the host protocol binaries, board code files and the example software from the User Part Development Package. This allows users to update multiple components within a single installation cycle.

The host-based software uses a three part release number in the form "Release x.y.z" to uniquely identify the software version. Furthermore, the host-based binaries that form part of this distribution have the same release number.

For example:

```
DSI gctload Release 6.5.0 (Build 1121)
Part of the Dialogic(R) DSI Development Package for Linux
Copyright (C) 1994-2012 Dialogic Inc. All Rights Reserved.
```

The DSI shared objects are located in sub-directories, '32' for the 32 bit libraries and '64' for the 64 bit libraries.

Installation of the software is described in more detail in the following topics:

- Installing Development Package for Linux
- Installing the DSI Source Device Drivers
- Support for Native SCTP
- Removing the Development Package for Linux
- RPM Creation

3.2.1 Installing Development Package for Linux

Install the Development Package as follows:

1) Remove any existing Development Package installation referring to Section 3.2.6 Removing the Development Package for Linux on page 27.
2) Login and switch to a user account with root privileges.
3) Create a new directory on the development system to act as the root directory for the software. This directory is referred to as the install directory.

```
mkdir /opt/DSI
cd /opt/DSI
```

4) Copy the dpklnx.Z file to the install directory. Take care to retain the Z extension (which identifies the file as a compressed file) and ensure binary file integrity is maintained.
5) Extract the files using the command:

```
tar --no-same-owner -zxvf dpklnx.Z
```

The following files (or similar) are extracted into the current working directory.

Table 2. Files Installed on a System Running Linux

Sub- Directory	File name	Description
	example_system.txt	Example system configuration file
	example_config.txt	Example protocol configuration file.
	dsictrl, dsistat,dsitrace, gctload, rsi, rsi_lnk, rsicmd s7_log, s7_mgt, s7_play, sctp, sctpd, sctpn, ssdh, ssdl, ssdm, ssds, tempmon, tick, tim, etc	General purpose DSI host utilities documented within this manual
	dsa, dtc, hstmgr, rmm, , ssm, etc	Specialized host utilities documented elsewhere
	ctu, intu, istr, istu, mtr, mtu, ttu, upe, etc	Ready built versions of the examples contained in the User Part Development Package.
	dsi-mibs.zip	Dialogic® DSMI SNMP MIBs distributed as a compressed ZIP file
32	libgctjni.so, libgctlib.so, libgctlib.so.1, libgctlib.so.1.49.0, libin_api.so	Sub-directory containing 32 bit shared object libraries
64	libgctjni.so, libgctlib.so, libgctlib.so.1, libgctlib.so.1.49.0, libin_api.so	Sub-directory containing 64 bit shared object libraries
DC	monitor.dc3, ss7.dc3, ss7.dc4, ss7.dc6, ss7.dc6amc, ss7.dc7, ss7mcd.dc6 etc	Sub-directory containing downloadable Code Files
HSTBIN	dmr, inap, is41, isup, m2pa, m3ua, map, mst, mtp3, sccp, sua, tcap, tup etc.	Sub-directory containing Host Protocol Binaries
INC		Sub-directory containing header files for use with user's application.
JAVA	dmrApi.jar, dmtrCmds.jar, dms.jar, gctApi.jar, etc	Sub-directory containing Java files.
SPCI_CPM_DRIVER		Sub-directory containing driver source code and makefiles for SPCI boards.
SS7HD_DRIVER		Sub-directory containing driver source code and makefiles for SS7HD boards.
SS7LD_DRIVER		Sub-directory containing driver source code and makefiles for SS7LD boards.
SS7MD_DRIVER		Sub-directory containing driver source code and makefiles for SS7MD boards.
UPD		Sub-directory containing the User Part Development package.

6) Add the following lines to the file `/etc/ld.so.conf` indicating the path to the shared object:

```
/opt/DSI/32
/opt/DSI/64
```

7) Run ldconfig to configure the dynamic linker's run time configuration.

```
ldconfig -v
```

A series of links will be configured, similar to the following example (the name of libgctlib.so will reflect the current version number of the shared object library):

```
/opt/DSI/32:
        libgctjni.so -> libgctjni.so
        libin_api.so -> libin_api.so
        libgctlib.so.1 -> libgctlib.so.1.49.0
/opt/DSI/64:
        libgctjni.so -> libgctjni.so
        libin_api.so -> libin_api.so
        libgctlib.so.1 -> libgctlib.so.1.49.0
```

8) When using Java-based APIs to connect into the GCT environment then the location of the libgctjni.so can optionally be set using the LD_LIBRARY_PATH environment variable. The syntax of the command to set this variable will vary depeding on the system.

```
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/opt/DSI/32
```

The user can alternatively set the library path each time the java command is run. For example, to run the example jar file /opt/DSI/JAVA/dtr.jar:

```
java -Djava.library.path=/opt/DSI/32 -jar /opt/DSI/JAVA/dtr.jar
```

Note: For 64 bit systems replace /opt/DSI/32 with /opt/DSI/64.

9) Before proceeding, verify that the shared object is installed correctly using the gctload command with the –v option to print out version information.

Note: The DSI binaries require the 32 bit run-time libraries (libc.so). to be installed. Some 64 bit Linux distributions only install 64 bit run-time libraries by default. Refer to your distribution's documentation for instructions on how to install the 32 bit run-time libraries.

```
gctload –v
```

If the shared object is not correctly installed then an error message will be printed out instead. E.g:

```
./gctload: error while loading share libraries: libgctlib.so.1: cannot
open shared object file: No such file or directory
```

10) If SIGTRAN software is going to be used then see section 3.2.3 Support for SIGTRAN SCTP under Linux for further details.

If using the SCTPD binary then change the privileges for the binary as follows:

```
chown root sctpd
chmod +s sctpd
```

If using the MST binary then, change the privileges as follows:

```
chown root ./HSTBIN/mst
chmod +s ./HSTBIN/mst
```

11) To reserve sufficient system resources, the Linux kernel parameters should be set as detailed in section 3.2.4 Adjusting Linux Kernel Parameters on page 26.

3.2.2 Building Device Drivers for DSI boards

Once the Development Package is installed, the device drivers for all DSI boards are contained in a per-board folder. If using a DSI board it is first necessary to build the device driver in your target environment.

Build scripts and installation scripts are supplied in the following locations:

Dialogic® DSI Network Interface Board Family	Sub-directory	Build Script	Install Script
SPCI	SPCI_CPM_DRIVER	build_spci_cpm.sh	install_spci_cpm.sh
SS7HD	SS7HD_DRIVER	build_ss7hd.sh	install_ss7hd.sh
SS7LD	SS7LD_DRIVER	build_ss7ld.sh	install_ss7ld.sh
SS7MD	SS7MD_DRIVER	build_ss7md.sh	install_ss7md.sh

To build the driver, run the appropriate script. The build script assumes that a suitable environment for building Kernel modules is available. This must include the appropriate Kernel include files being found at: /usr/src/linux-`uname -r`/include (e.g., /usr/src/linux 2.6.5/include). If these are not found, the build will fail.

Note: When installing the Development Package in systems that include a DNIxxxxTEPE2HMP board it is important NOT to install the SS7LD device driver. The driver from the Dialogic® PowerMedia™ HMP 4.1 Linux (SU 151 or later) release includes a driver that also supports the SS7LD).

Some Linux installations do not create a system source directory with the required name; for example, some SMP kernels do not create a directory with the required smp suffix. If this is the case, then a softlink needs to be created to give an appropriate path to the system header files. For example:

```
cd /usr/src
ln –s linux-2.6.5 linux-2.6.5 smp
```

Some later versions of Linux use a revised format for the remap_page_range parameters (for example, Red Hat Linux Kernel Versions greater than 2.4.20 require this revised format). The build script supports an optional new_remap parameter. If this parameter is set, the compile uses the revised format.

The build script supports an optional clean parameter that removes the driver and all intermediate files.

Under some versions of Linux a warning similar to the following is generated which can safely be ignored:

```
warning: changing search order for system directory.
```

Once the driver has been successfully built, the appropriate install script should be invoked. This installs the device driver, automatically allocates the major device numbers, and creates the four appropriate device nodes.

Driver installation must be performed by a user with root privileges.

Correct loading of the device driver can be confirmed by looking in the system log. The system log is displayed using the command:

```
dmesg | more
```

Successful installation of the driver is indicated by the allocation of a device id. (Note that a Device Id will only be allocated if the target DSI board is present in the system when the driver is built).

Example output, using the SPCI board, is:

```
DSI SPCI Release 6.3.3 (Build 1071)
Part of the Dialogic(R) DSI Development Package for Linux
Copyright (C) Dialogic Corporation 2000-2011. All Rights Reserved.
Build options: NEW_DDK
Using major device number 253.
sptpci[0]: DPM offset adjustment 0x0
ACPI: PCI Interrupt 0000:08:02.0[A] -> GSI 18 (level, low) -> IRQ 185
  sptpci Device Id 0 @ Bus: 8 Device: 2 Function: 0
```

The install script supports an optional remove parameter. This causes the device driver to be removed and the device nodes to be deleted. For example:

```
install_*.sh remove
```

3.2.3 Support for SIGTRAN SCTP under Linux

The Development Package for Linux supports two different SCTP configurations:

The SCTP binary (in conjunction with the SCTPD binary) provides a complete implementation of the SIGTRAN SCTP protocol suitable for use with pre-2.6.16 kernels which do not support Native SCTP within the kernel.

The SCTPN binary (in conjunction with the native SCTP capability within Linux) provides a complete implementation of the SIGTRAN SCTP stack suitable for use on kernels that do support the Native SCTP capability.

Operation of the SCTPN binary in conjunction with the kernel SCTP implementation requires versions 2.6.16 or greater of the Linux kernel and 1.0.6 or greater of the lksctp-tools package. Please note that the lksctp-tools package may not be installed by default on some Linux distributions, in which case it must be installed manually.

Linux systems using (Security Enhanced Linux) SELinux or other firewalls may require further configuration to allow SCTP traffic to be sent and received.

To make use of the Native SCTP capability, the user should use the SCTPN binary instead of the binaries SCTP and SCTPD which are usually found in the SS7 Development Package installation directory (the recommended location is `/opt/DSI`).

Before starting the system, the sctp loadable kernel module will usually need to be inserted into the system. This can be done using the modprobe command:

```
modprobe sctp
```

On systems with Linux kernel version 2.6.16 or greater, adding the following lines to `/etc/modprobe.conf` will cause the system to insert the kernel module automatically on demand:

```
alias net-pf-10-proto-132 sctp
alias net-pf-2-proto-132 sctp
```

3.2.4 Adjusting Linux Kernel Parameters

To reserve sufficient system resources for the DSI inter-process communication mechanism to function correctly it is important to adjust certain kernel parameters to suitable values.

For linux, the **kernel.msgmnb** parameter usually needs to be adjusted.

kernel.msgmnb should be set to at least 12 times the total number of messages configured in the system.txt file (in addition to any requirements of other software making use of these resources).

Once system.txt is written, add together NUM_MSGS and NUM_LMSGS and multiply the result by 12. Add on any requirements from other software to determine the required setting for kernel.msgmnb.

Edit the `/etc/rc.local` (or distribution-specific equivalent) file to add the following line:

```
sysctl -w kernel.msgmnb=62400
```

For Linux, the kernel.msgmni parameter controls the number of message queues supported.

The kernel.msgmni parameter should be set to the number of module queues defined in system.txt with the LOCAL command (in addition to any requirements of other software making use of these resources).

Edit the /etc/rc.local (or distribution-specific equivalent) file to add the following line:

```
sysctl -w kernel.msgmni=256
```

When run, GCTLOAD will attempt to allocate the maximum number of system resources in order to verify, as far as possible, that the kernel parameters have been correctly adjusted.

3.2.5 Using 64-bit Linux Applications

The Development Package includes both 32-bit and 64-bit GCTLIB shared object files, allowing 64-bit user applications to co-exist with 32-bit DSI software. Both libraries share the same naming convention (libgctlib.x.y.z .), with the 32-bit library stored in the 32 directory and the 64-bit library stored in the 64 directory.

Note: The 'x.y.z' of 'libgctlib.so.x.y.z' refers to the GCTLIB shared object's major, minor and release version numbers.

Users should update the target system's run-time linker's shared object search paths to include the paths to the 32-and 64-bit GCTLIB shared libraries as required.

To create a 64-bit application, users must ensure that their application code does not access the 'next' field in the HDR structure of a message. This field is called 'hdr.next' in a 32-bit environment and 'hdr.next_ref' in a 64-bit environment. Any existing application code that made use of this field needs to be removed.

To build a 64-bit application, all Makefiles and/or IDE configurations need to be modified to define DSI_64BIT, for example, by editing the User Part Development Package's makdefs.mak to:

```
DKDEFINES = -DLINT_ARGS -DIN_LMSGS -DDSI_64BIT
```

All 64 bit user applications should be linked against the 64 bit version of GCTLIB which is installed by default in the following location:

```
64/gctlib.so.x.y.z.
```

3.2.6 Removing the Development Package for Linux

Prior to installing a new version of the Development Package for Linux, the previous version should be removed. This can be achieved using the following procedure (assuming the user logs on as root):

1) Back-up any license files and user configuration files (eg. system.txt, config.txt, *.ms7 scripts etc) so that these may be re-installed after the new installation has been completed.

2) Delete both the installed files and the directory `/opt/DSI`.

3) The file `/etc/ld.so.conf` should be edited and the line indicating the path to the shared object should be removed

   For 32 bit systems this should be: `/opt/DSI/32`

   For 64 bit systems this should be: `/opt/DSI/64`

4) Then `ldconfig` should be run to re-configure the dynamic linker's run time configuration.

5) Reboot the target machine.

3.2.7 RPM Creation

The Development Package provides support for the generation of RPM (RedHat Package Management) packages for Linux kernels from V2.6.

RPM Creation Instructions

Note: The 'rpmbuild' package must be installed before the following steps can be performed.

A number of RPM packages will be created from the Development Package. The RPM packages are created by executing the following steps:

1. Select a directory to be used when creating the RPM packages.

For this example, "`/var/tmp/dpk/rpm`" is used.

2. Create a file called "`.rpmmacros`" in the user account's home directory and enter the location of the directory from step 1:

```
%_topdir /var/tmp/dpk/rpm
```

3. Prepare the RPM directory:

```
mkdir -p /var/tmp/dpk/rpm/{BUILD,RPMS,SOURCES,SPECS,SRPMS}
```

4. Copy the dpklnx.Z file to the user account's home directory. Take care to retain the Z extension (which identifies the file as a compressed file) and ensure binary file integrity is maintained.

5. Execute rpmbuild:

```
rpmbuild -tb dpklnx.Z
```

6. For 32bit operation systems, the RPM packages are stored in: `/var/tmp/dpk/rpm/RPMS/i386/.`

For 64bit operation systems, the RPM packages are stored in:
`/var/tmp/dpk/rpm/RPMS/x86_64/`

For example:

```
ls /var/tmp/dpk/rpm/RPMS/<ARCH>/
ss7dpk-5.08-1.<ARCH>.rpm
ss7dpk-devel-5.08-1.<ARCH>.rpm
ss7dpk-debuginfo-5.08-1.<ARCH>.rpm
ss7dpk-kmod-5.08-1.2.6.9_34.EL.<ARCH>.rpm
```

Where `<ARCH>` is i386 for 32bit operation and x86_64 for 64 bit operation systems.

Note: Device driver binaries will be built as rpmbuild is run. Therefore, it is necessary for the machine on which rpmbuild is run to share the same kernel version as the machine on which the RPM packages will be installed.

RPM Packages

The following packages are created:

`ss7dpk-<DPK>.<ARCH>.rpm` Run-time files, including binaries, GCT run-time shared library and system.txt and config.txt configuration files.

`ss7dpk-devel-<DPK>.<ARCH>.rpm` Development Package development files, including header files and GCT link-time shared library.

`ss7dpk-kmod-<DPK>-<KERNEL>.<ARCH>.rpm` Signaling boards device drivers binaries.

`ss7dpk-debuginfo-<DPK>.<ARCH>.rpm` RPM build artifact, not required.

Using the RPM Management Tool

The RPM management tool, "rpm", is used to maintain packages on a target system. Documentation on how to use the "rpm" tool is available from www.rpm.org.

Common tasks using the rpm utility include:

1. Installation of an RPM package:

```
rpm -i <package_name>
```

2. Removal of an installed RPM package:

```
rpm -e <package_name>
```

3. Upgrading an installed RPM package:

```
rpm -U <package>
```

4. List all RPM packages on a system:

```
rpm –qa
```

3.3 Software Installation for Solaris

Installation of the software is described in more detail in the following topics:

- Installing the Development Package for Solaris
- Removing the Development Package for Solaris
- Solaris 9 - Interface Name Checking
- Solaris 10 - Additional Commands
- Choice and configuration of SCTP module
- Non-serviced interrupts reports

The Development Package for Solaris includes, in a single distribution, the software required by users of the Dialogic® DSI SS7 Boards and Dialogic® DSI Protocol Stacks.

The Development Package now includes the host protocol binaries, board code files and the example software from the User Part Development Package. This allows users to update multiple components within a single installation cycle.

The host-based software uses a three part release number in the form "Release x.y.z" to uniquely identify the software version. Furthermore, the host-based binaries that form part of this distribution have the same release number.

For example:

```
DSI gctload Release 6.2.9 (Build 1055)
Part of the Dialogic(R) DSI Development Package for Solaris(SPARC)
Copyright (C) Dialogic Corporation 1994-2010. All Rights Reserved.
```

The consolidated Development Packages for Solaris are distributed within two compressed 'tar' archive files:

dpksparc.tar.gz - Solaris Packages for Solaris-SPARC

dpkx86.tar.gz - Solaris Packages for Solaris-x86

Each distribution contains two Solaris packages:

dsidpk - DSI Development Package

dsidrv - DSI Network Interface Board Driver Package

All users need to install the 'dsidpk' package, whereas only users of signaling boards will need to install the 'dsidrv' package. Both packages contain support for 32 bit and 64 bit systems and the installation process selects the appropriate package for the target system.

These files can be downloaded from the Dialogic website. See Section 1.1 Related Documentation.

3.3.1 Installing the Development Package for Solaris

The Solaris package installation steps are:

1) Backup any license files and user generated configuration files (e.g. system.txt, config.txt, *.ms7 scripts etc) so that they may be re-installed after the new installation is complete.

2) Remove any existing Development Package installation referring to Section 3.3.9 Removing the Development Package for Solaris on page 35.

3) Select the correct Solaris distribution and extract the Solaris package files:

For SPARC systems:

```
gzip -d dpksparc.tar.gz
tar -xf dpksparc.tar
```

For x86 systems:

```
gzip -d dpkx86.tar.gz
tar -xf dpkx86.tar
```

4) While logged-on as 'root', install the extracted Solaris packages:

```
pkgadd -d dsidrv
pkgadd -d dsidpk
```

Note: 'dsidrv' is only necessary for users requiring board-level drivers.

The following files (or similar) are transferred into the root installation directory (`/opt/DSI`).

Table 3. Files Installed on a System Running Solaris

Sub- Directory	File name	Description
	example_config.txt	Example protocol configuration file.
	example_system.txt	Example system configuration file
	dsictrl, dsistat, dsitrace, gctload, s7_log, s7_mgt, s7_play, rsi, rsi_lnk, rsicmd, sctp, sctpd, sctpn, ssdh, ssdl, ssdm, ssds, tempmon, tick, tim, etc	General purpose DSI host utilities documented within this manual
	dsa, dtc, hstmgr, mbm, rmm, etc	Specialized host utilities documented elsewhere
	ctu, intu, istr, istu, mtr, mtu, ttu, upe, etc	Ready built versions of the examples contained in the User Part Development Package.
	dsi-mibs.zip	Dialogic® DSMI SNMP MIBs distributed as a compressed ZIP file
32	libgctjni.so, libgctlib.so , libgctlib.so.1, libgctlib.so.1.49.0, libgctlib_nomco.so.1.49.0, libin_api.so	Sub-directory containing 32 bit shared object libraries
64	libgctlib.so, libgctlib.so.1, libgctlib.so.1.49.0, libgctlib_nomco.so.1.49.0, libin_api.so	Sub-directory containing 64 bit shared object libraries
DC	monitor.dc3, ss7.dc3, ss7.dc4, ss7.dc6, ss7.dc7, ss7mcd.dc6, etc	Sub-directory containing downloadable Code Files

Sub- Directory	File name	Description
HSTBIN	dmr, inap, is41, isup, m2pa, m3ua, map, mst, mtp3, sccp, sua, tcap, tup etc.	Sub-directory containing Host Protocol Binaries
INC		Sub-directory containing header files for use with user's application.
JAVA	dmrApi.jar, dmtrCmds.jar, dms.jar, gctApi.jar, etc	Sub-directory containing Java files.
UPD		Sub-directory containing the User Part Development package.

Note: To reserve sufficient system resources, the Solaris System Resource parameters should be set as detailed in Section 3.3.5 Tuning Solaris System Resource Parameters.

3.3.2 Solaris 9 - Interface Name Checking

To use the package under Solaris 9, interface name checking must be disabled. (otherwise the device driver may not start correctly) This is achieved by adding the following line to the /etc/system file:

```
set sunddi_netifname_constraints=0
```

3.3.3 Solaris 10 –User Account Permissions

On Solaris 10, all non-root user accounts must be updated to access the DSI Signaling board.

For example, the command (which must be executed by root) to update 'dsiuser' is:

```
usermod -K defaultpriv=basic,net_rawaccess dsiuser
```

3.3.4 Installation of SIGTRAN support for Solaris

The Development Package for Solaris supports two different SCTP configurations:

The SCTP binary (in conjunction with the SCTPD binary) provides a complete implementation of the SIGTRAN SCTP protocol suitable for use on Solaris 9.

The SCTPN binary (in conjunction with the native SCTP capability within Solaris) provides a complete implementation of the SIGTRAN SCTP stack suitable for use on Solaris 10 or later.

Note: If using the SCTPN binary, then this must be running with superuser privileges. From DSI Development Package for Solaris Release 5.2.1 this is the default setting, for earlier releases this can be achieved by ensuring the binary is owned by 'root' and have the binary file setuid bit set. If these are not set then the binary will not be able to modify all of the appropriate kernel settings such as timers.

3.3.5 Tuning Solaris System Resource Parameters

When using Solaris, it is essential to configure the kernel such that sufficient resources are made available for inter-process communications. Failure to complete this step may cause the system to halt.

Note: The Sun document: 'System Administration Guide: Solaris Containers-Resource Management and Solaris Zones' (http://www.sun.com/documentation) is the designated reference relating to Solaris resource management.

Three Solaris resources parameters need to be set (the names of these parameters differ between Solaris 9 and Solaris 10 - both naming conventions are shown for clarity) as follows, although users should also take into account other applications that may require these resources:

Solaris 10 Solaris Resource Name	Solaris 9 Old /etc/system name	Setting
process.max-msg-messages	msginfo_msgtql	Set to a value greater than or equal to the sum of NUM_MSGS + NUM_LMSGS
process.max-msg-qbytes[2]	msginfo_msgmnb	Set to a value greater than or equal to the number of NUM_MSGS + NUM_LMSGS multiplied by 12.
project.max-msg-ids	msginfo_msgmni	Set to a value greater than or equal to the number of LOCAL message queues.

3.3.6 Creating a Solaris 'project' to tune System Resource parameters

Solaris projects provide a mechanism for grouping multiple configuration options. They also provide an administrative identifier for related work.

1) The following example creates a new project (gctenv), adds the user (gctuser) to the project, and modifies the projects attributes. Create a new project, gctenv

```
projadd gctenv
```

2) Add the user, gctuser, to the project:

```
projmod –a –U gctuser gctenv
```

3) Modify the projects' attributes according to the size of the GCT resources.

In this example, the target system will use 20 message queues (20 instances of LOCAL in system.txt) and 10000 messages and 1000 long messages – giving a total of 11000 messages. A 10% 'margin of error' has been added to each resource value:

```
projmod -a -K "process.max-msg-messages=(priv,12100,deny)" gctenv
projmod -a -K "project.max-msg-ids=(priv,22,deny)" gctenv
projmod -a -K "process.max-msg-qbytes=(priv,145200,deny)" gctenv
```

Note: Each projmod command is a single line.

4) Make the project gctenv the default project for user gctuser. As root, edit /etc/user_attr and add:

[2] The process.max-msg-qbytes/msginfo_msgmnb values specified are the System V (SYS V) Interprocess Communications (IPC) values required for the correct operation of DSI Software Environment. Other application software may use the SYSV IPC resources and, therefore, their configuration requirements must be added to the process.max-msg-qbytes/msginfo_msgmnb total.

```
gctuser::::project=gctenv
```

Note: There are four (4) colons between 'gctuser' and 'project'.

5) Login as `gctuser` and verify the correct default project

```
id -p
uid=100(gctuser) gid=1(other) projid=100(gctenv)
```

3.3.7 Using 64-bit Solaris Applications

The Development Package includes both 32-bit and 64-bit GCTLIB shared objects (shared libraries), allowing 64-bit user applications to co-exist with 32-bit DSI software. Both libraries share the same naming convention (libgctlib.x.y.z .) with the 64-bit library stored in the 64-directory.

Note: The 'x.y.z' of 'libgctlib.so.x.y.z' refers to the GCTLIB shared object's major, minor and release version numbers.

To create a 64-bit application, users must ensure that their application code does not access the 'next' field in the HDR structure of a message. This field is called 'hdr.next' in a 32-bit environment and 'hdr.next_ref' in a 64-bit environment.

To build a 64-bit application, all Makefiles and/or IDE configurations need to be modified to define DSI_64BIT, for example, edit the User Part Development Package's makdefs.mak to:

```
DKDEFINES = -DLINT_ARGS -DIN_LMSGS -DDSI_64BIT
```

64 bit user applications should be linked against the 64 bit version of GCTLIB. This is installed by default in the following location:

```
/opt/DSI/64/gctlib.so.x.y.z.
```

3.3.8 Avoiding "Non-serviced interrupt" reports

Some systems exhibit issues due to non-serviced interrupts being reported by the system. The issue can result in large numbers of event reports that can impact the system performance.

The DSI Board drivers included in this package include an optional work-around to eliminate these issues.

To enable this functionality, the following line must be added to the /etc/system file:

DSI Network Interface Board	
SPCI2S or SPCI4	`set septel:spt_claimint=1`
SS7HD	`set ss7hd:ss7hd_claimint=1`
SS7MD	No change required

Note: The system has to be rebooted to force the change to take effect.

3.3.9 Removing the Development Package for Solaris

Before removing the Development Package users should take a back-up of any user configuration files (eg. system.txt and config.txt) so that these may be re-installed after the new installation has been completed.

The Development Package for Solaris can be removed using the package removal utility as follows Removal must be performed by a user with Administrator privileges:

```
pkgrm dsidpk
pkgrm dsidrv
```

The Solaris package removal utility (pkgrm) then prompts for further input.

On successful completion of the procedure, the following message is displayed and the user should reboot the system:

```
Removal of <dsidpk / dsidrv> was successful.
```

3.4 Software Installation for Windows

The Development Package for Windows is distributed electronically as a download from the Dialogic website. See Section 1.1 Related Documentation. The distribution is in the form of a single self extracting binary named DPKWIN.EXE. This binary can be run directly from a hard disk.

Installation and removal of the software is described in more detail in the following topics:

- Installing Development Package for Windows
- Starting the Windows Device Driver
- Removing Development Package for Windows

3.4.1 Installing Development Package for Windows

If the development package is to be used with a Dialogic® DSI Network Interface Board, then the board must be installed before installation of the Development Package such that the driver is correctly loaded.

1) Backup any user generated configuration files (e.g. system.txt, config.txt, *.ms7 scripts etc) so that they may be re-installed after the new installation is complete.
2) Remove any existing Development Package installation referring to section 3.4.6 Removing Development Package for Windows on page 41.
3) The installation must be performed by a user with Administrator privileges. Before performing the installation, close all other applications.
4) To perform the installation, run the self-extracting binary DPKWIN.EXE. The installation procedure will ask you to select the driver to be installed; select the required driver. You may also select an installation directory. The default directory is C:\DSI. If required, the default directory can be modified.

Note: When installing the Development Package in systems that include a DNIxxxxTEPE2HMP board it is important NOT to install the SS7LD device driver. The driver from the Dialogic® PowerMediaTM HMP 3.0 Windows (SU 347 or later) release includes a driver that also supports the SS7LD).

The following files (or similar) are transferred to the installation directory.

Table 4. Files Installed on a System Running Windows

Sub- Directory	File name	Description
	example_system.txt	Example system configuration file
	example_config.txt	Example protocol configuration file.
	ddinst.exe	Device driver installer (common between SPCI and SS7HD).
	gctload.exe, gctserv.exe, servcfg.exe, tick.exe, tim.exe, s7_mgt.exe, s7_log.exe, s7_play.exe, dsictrl.exe, dsitrace.exe, dsistat.exe, sctp.exe, sctpd.exe, ssdh.exe, ssds.exe, etc	General purpose DSI host utilities documented within this manual

Sub- Directory	File name	Description
	dtc.exe, rmm.exe, rsi.exe, rsi_lnk.exe, rsicmd.exe, txa.exe, etc	Specialized host utilities documented elsewhere
	ctu, intu, istr, istu, mtr, mtu, ttu, upe, etc	Ready built versions of the examples contained in the User Part Development Package.
	dsi-mibs.zip	Dialogic® DSMI SNMP MIBs distributed as a compressed ZIP file
32	gctlib.dll, gcltlib.lib	Sub-directory containing 32 bit shared object libraries
64	gctlib.dll, gctlib.lib	Sub-directory containing 64 bit shared object libraries
DC	ss7.dc3, ss7.dc4, etc	Sub-directory containing downloadable Code Files
HSTBIN	Inap.exe, is41.exe, isup.exe, m2pa.exe, m3ua.exe, map.exe, mst.exe, mtp3.exe, sccp.exe, sua.exe, tcap.exe, tup.exe, etc.	Sub-directory containing Host Protocol Binaries
INC		Sub-directory containing header files for use with user's application.
UPD		Sub-directory containing the User Part Development package.
SPCIDVR		Sub-directory containing 32 and 64 bit device drivers for DSI SPCI boards
SS7HDDVR		Sub-directory containing 32 and 64 bit device drivers for DSI SS7HD boards
SS7LD		Sub-directory containing 32 and 64 bit device drivers for DSI SS7LD boards.

The setup program may request a reboot of the target machine when it has finished installing the package. If requested, then the machine should be allowed to reboot.

The files the user needs to use have been installed in the installation directory. It is recommended that the user not modify any files in this directory, but instead create a working directory into which all the necessary files are copied.

3.4.2 Starting the Windows Device Driver

Under Windows, a plug and play driver will be installed for DSI boards when the new development package is installed on the system. The system will automatically detect any of the DSI Network Interface Boards and configure the driver for these boards.

To confirm the correct behavior, proceed as follows:

1) Choose Start -> Control Panel to open the Control Panel dialog.

2) Double-click on the Administrative Tools icon, then double-click the Computer Management icon to open the Computer Management window.

3) Click on the Device Manager tree item to display a tree of devices in the right window pane.

4) Check for the appropriate Board device under Dialogic DSI SS7 Boards. If the driver is not correctly installed, there will be a question mark (?) or an exclamation mark (!) before the SS7 Board item.

3.4.3 Additional steps using Windows 7

For Windows 7 users, the following additional steps are necessary to configure the DSI software environment:

1) Open up the <DPK> folder (normally 'c:\DSI' on a Windows system) so that you have access to the root of the installation where "gctload.exe" etc is located

2) Right click on "gctload.exe" and select 'properties'

3) A new window will pop-up

4) Select the 'Compatibility' tab

5) Now click on the "Run as Administrator option"

6) Click on Apply / OK and then close

3.4.4 Running software as a Windows Service

The Development Package for Windows can be configured to allow it to be automatically executed at system initialization. This is achieved by running it via a Windows Service. Running as a Service allows for the automatic invocation of gctload at system boot; it also allows the stopping and restarting of gctload via a standard programming interface and additionally provides a mechanism for remote restarting of the DSI software.

This functionality utilizes the following two executables which are part of the Development Package for Windows.

gctserv.exe - Service executable.

servcfg.exe - Service configuration and installation tool.

Installing a Service

Note: For Windows 7, all commands must be run from a command shell (cmd.exe) which has been run with 'Administrator' privileges. To select 'Administrator' privileges run Windows Explorer, find the 'cmd.exe' file, right click on 'cmd.exe' and select 'Run as administrator'.

Before the Service can be installed, the executable must be copied to the appropriate directory of the Windows installation.

For 32-bit Windows:

```
copy C:\DSI\gctserv.exe %WINDIR%\system32
```

For 64-bit Windows:

```
copy C:\DSI\gctserv.exe %WINDIR%\syswow64
```

For 32-bit operating systems the 32-bit gctlib.dll file must also be copied to the %WINDIR%\system32 directory.

```
copy C:\DSI\32\gctlib.dll %WINDIR%\system32
```

For 64-bit operating systems copy the 32-bit gctlib.dll into the SYSWOW64 directory as follows (this DLL will be used by the WOW emulator when running any of the standard (32-bit) binaries that are part of the development package):

```
copy C:\DSI\32\gctlib.dll C:\%WINDIR%\SYSWOW64
```

The installation is performed using the executable servcfg.exe. This installation must be performed by a user with Administrator privileges.

When installed, the Service is identified by the name "Dialogic® DSI Startup Service" within the 'services' utility.

The command line format for Service installation are:

32-bit Windows:

```
servcfg.exe install %WINDIR%\system32\gctserv.exe <gctload> <system.txt>
<start_dir>
```

64-bit Windows:

```
servcfg.exe install %WINDIR%\syswow64\gctserv.exe <gctload> <system.txt>
<start_dir>
```

Where

<gctload> is the full pathname for the gctload executable, and

<system.txt> is the pathname for the system configuration file.

<start_dir> is the directory in which the Service is started. All files referenced by the gctload executables (including the system.txt and all executables specified within) must be specified with pathnames relative to this directory (or as absolute path names).

For example, if **system.txt** is present in the **c:\DSI** directory, the following command would be used to configure the Service:

32-bit Windows:

```
servcfg.exe install %WINDIR%\system32\gctserv.exe c:\DSI\gctload.exe
system.txt c:\DSI
```

64-bit Windows:

```
servcfg.exe install %WINDIR%\syswow64\gctserv.exe c:\DSI\gctload.exe
system.txt c:\DSI
```

When the Service is installed, by default, the startup mode is set to 'manual'. To cause the Service to be automatically invoked at boot time it must be explicitly configured to 'automatic' mode. This is achieved by running the Services tool and setting the startup option to "automatic".

Under Windows Server® 2008 and Windows Server® 2008 R2 operating systems the Services tool can be found under Control Panel -> Administrative Tools -> Services.

Under the Windows® 7 operating system the Services tool is located under Control Panel -> System and Security -> Administrative Tools -> Services.

Uninstalling a Service

The Windows Service is also removed using the executable servcfg.exe using the syntax given below and can be removed from the system32 directory as follows:

```
servcfg.exe remove
del servcfg.exe
```

Running the Service manually

The Service is started manually using the "Services" tool.

Select the required Service ("Dialogic DSI Startup Service") and start the Service using the start icon. When the Service has been successfully started, the displayed status of the Service is "started".

The Service is stopped manually using the "Services" tool (using the button labeled "stop" or the stop icon). When the Service has been successfully stopped, the displayed status of the Service is "stop".

3.4.5 Using 64-bit Windows Applications

The Dialogic® DSI Development Package for Windows offers support for both 32-bit and 64-bit applications. For 32-bit applications, users should use the gctlib.dll library whilst for 64-bit applications the gctlib.dll library (which is shipped in the DSI\64 directory of the distribution) should be used.

Having installed the DSI Development Package for Windows, users wishing to use 64-bit applications need to perform the following additional steps (which assume the default installation path, C:\DSI).

Copy the (32-bit) gctlib.dll into the SYSWOW64 directory as follows (this DLL will be used by the WOW emulator when running any of the standard (32-bit) binaries that are part of the development package):

```
Copy C:\DSI\32\gctlib.dll C:\WINDOWS\SYSWOW64
```

Copy the (64-bit) gctlib.dll file into the system32 directory as follows. This DLL will be used by the user's (64-bit) application (the system32 directory is the default location for 64-bit binaries in a 64-bit system, despite its name).

```
Copy C:\DSI\64\gctlib.dll C:\WINDOWS\system32
```

For correct operation of 64-bit applications, it is essential that the user does not access the 'next' field in the HDR structure of a message. This field is called 'hdr.next' in a 32-bit environment and 'hdr.next_ref' in a 64-bit environment. Any existing application code that made use of this field needs to be removed.

All Makefiles and/or IDE configurations need to define DSI_64BIT, for example, edit the User Part Development Package's makdefs.mnt to:

```
DKDEFINES = -DLINT_ARGS -DIN_LMSGS -DDSI_64BIT
```

All 64 bit user applications should be linked against the 64 bit version of GCTLIB. This is installed by default in the following location:

```
C:\DSI\64\gctlib.lib
```

3.4.6 Removing Development Package for Windows

Prior to installing a new version of the Development Package for Windows, the previous package must be removed as follows. This procedure requires a user with Administrator privilege.

1) Back-up any license files and user configuration files (eg. system.txt and config.txt) so that these may be re-installed after the new installation has been completed.
2) Select the Control Panel (Start → Settings → Control Panel).
3) Select "Add/Remove Programs".
4) Scroll down the devices and select "Dialogic® DSI SS7 Development Package" and select "Remove".
5) When package removal is confirmed, restart the target machine.

4 Configuration and Operation

Before attempting software configuration, you should gain an understanding of the flexibility of the protocol stack, the run-time options that exist and the mechanisms that are used to select specific features. This Section gives an overview of these options. You should also read Section 2 Basic Concepts that describes the basic principles of modules and message passing.

This section provides information about:

- Selecting the System Architecture
- Creating the System Configuration File (system.txt)
- Creating the Protocol Configuration File (config.txt)
- Executing the Software
- Developing a User Application

4.1 Selecting the System Architecture

This section describes both board-based systems and SIGTRAN software-based systems. Both types of systems are supported by the Dialogic® DSI Protocol stack and the choice will usually be determined by the application and type of connection required.

4.1.1 TDM Board Systems

The Dialogic® DSI Protocol Stack software running on the board communicates with an application running on the host computer. The physical interface to the board uses the PCI bus. Communication with the board is handled by a device driver and message passing to and from the board is managed by the board management and interface process (ssdx, sometimes generically referred to as ssd) that runs on the host computer.

In addition to running the application on the host, the user may, depending on the Dialogic® DSI Network Interface Board, the size of the overall system and the network topology, choose to run some of the SS7 protocol modules on the host. See Section 4.1.4 Protocol Modules for more information. In such cases, the interface between the application and the SS7 protocol software remains identical. This allows for easy migration from a small system contained on a single board to a large system distributed over many boards with minimal changes to the application.

Table 5. Practical System Configurations for Telephony Systems

	Small System	Medium System	Large System
Board Support	SPCI (see note), SS7HD	SPCI, SS7HD	SPCI. SS7HD and SS7MD
Software running on board	MTP2 MTP3 ISUP / TUP / SCCP / TCAP / MAP / INAP / IS41	MTP2 MTP3	MTP2
Software running on Host CPU	User Application	ISUP / TUP / SCCP / TCAP / MAP / INAP / IS41 User Application	MTP3 ISUP / TUP / SCCP / TCAP / MAP / INAP / IS41 User Application

	Small System	Medium System	Large System
Number of boards	Single	Single board with signaling (although additional boards may be used to support voice only)	Multiple
Description	Suitable for single board solutions where the user wishes to maximize the available host processing power.	Suitable for single board solutions where the user wishes to make use of a high performance host to improve system throughput. Can also be used to support more complex protocol configurations.	Suitable for systems requiring larger numbers of links and systems that require distribution of MTP over multiple boards.

Note: The SPCI board only supports the use of MTP2, MTP3, ISUP and TUP protocols running on the board.

In board-based systems, the board management and interface process (ssd) is required to run on the host machine. The ssd process handles message transfer between the host and the board using the device driver.

4.1.2 SIGTRAN Systems

A software-only architecture may be configured using the appropriate Dialogic® DSI SIGTRAN modules in place of MTP2 and/or MTP3. These modules provide the same interface to upper protocol modules and use the services of SCTP to transport SS7 signaling reliably over IP.

SS7 Hosts may connect to SIGTRAN M3UA or M2PA Signaling Gateways such as the Dialogic® DSI Signaling Servers. Peer to peer connections between hosts using M2PA or M3UA are also supported.

The common interfaces presented to the upper layers enable applications to be easily ported between hardware and software-only architectures. The product designations are as follows:

- Dialogic® DSI M2PA – MTP2 Peer to Peer Adaptation Layer
- Dialogic® DSI M3UA – MTP3 User Adaptation Layer
- Dialogic® DSI SUA – SCCP User Adaptation Layer

As described in the installation section there are two choices of SCTP module – SCTP (used with SCTPD) and SCTPN which utilizes the 'native' SCTP stack provided by the host Operating System kernel.

Table 6. System Configurations for SIGTRAN Telephony Systems

	M2PA System	M3UA System	SUA System
Software running on Host CPU	SCTP / SCTPN M2PA MTP3 ISUP / TUP / SCCP / TCAP / MAP / INAP / IS41 User Application	SCTP / SCTPN M3UA ISUP / TUP / SCCP / TCAP / MAP / INAP / IS41 User Application	SCTP / SCTPN SUA TCAP / MAP / INAP / IS41 User Application
Description	Suitable for SIGTRAN solutions where MTP2 and below are replaced with an IP based solution. Useful when migrating an existing TDM system to IP	Suitable for SIGTRAN solutions where MTP3 and below are replaced with an IP based solution. M3UA provides the same interface as MTP3 to the upper layer modules	Suitable for SIGTRAN solutions where SCCP and below are replaced with an IP based solution.
Usage	Provides the ability to connect the Host Application to 8 Signaling Gateways or Remote Point Codes.	Provides the ability to connect the Host Application to 256 Signaling Gateways or IPSPs (remote hosts) and route to 256 remote Point Codes.	Provides the ability to connect the Host Application to 4 Signaling Gateways or 32 IPSPs (remote hosts) and route to 256 remote Point Codes.

4.1.3 Diameter Systems

Diameter systems may be set-up using the software-only architecture in the Dialogic® DSI Diameter Stack in conjunction with the SIGTRAN SCTP layer all of which are included in the DSI Development Package for Linux or Solaris. The Dialogic® DSI Diameter Stack includes a Diameter Module (DMR) implementing the core base protocol functionality and is supported by a Functional API library to aid application development.

Table 7. System Configurations for Diameter Systems

	Diameter System
Software running on Host CPU	SCTPN DMR (Diameter Module) User Application - built using the Diameter Functional API which is part of the Dialogic® DSI Diameter Stack).
Description	Suitable for Diameter solutions in either client or server configurations.
Usage	Provides the ability to connect the Host Application to Peer Diameters nodes.

4.1.4 Protocol Modules

The selection of which protocol modules to run on the host is made by editing the system.txt configuration file.

Some SS7 protocol modules can be run on either the host machine or on DSI Network Interface Boards. The options available for each individual board are described in the appropriate Programmer's Manual.

The user then runs the gctload program that reads the system configuration parameters from the system.txt configuration file and starts the selected processes bringing the system into operation. For further details on the operation of the gctload program, refer to 7.1 gctload.

Table 8 shows processes and utilities for use on the host that are included in the distribution.

Note: s7_mgt, s7_log and s7_play are optional utilities. A user may choose to implement the functionality provided by these utilities in their own applications.

Table 8. System Host Utilities

Process or Utility	Purpose
gctload	Process to initialize the system environment and start all other related processes running on the host, deriving the configuration from a text file (system.txt).
s7_log	Utility process to allow messages received from the protocol stack to be logged to a text file. This is useful for diagnostic purposes when getting started. Refer to "s7_log" for more information.
s7_mgt	Process to perform one time protocol configuration for all protocol modules, deriving the configuration parameters from a text file (config.txt). This process is optional. As an alternative to using it, the user may elect to perform protocol configuration by sending messages directly to the other modules in the system. Refer to the appropriate Programmer's Manual for information on configuration using discrete messages.
s7_play	Utility process used to generate messages from a text file and send them into the system. This is useful for diagnostic purposes when getting started. Refer to "s7_play" for more information.
ssdx	Process to interface with the device driver for passing messages to and from the board(s) and for downloading software to the board(s). Only required for TDM systems. Note: This process is referred to in a generic manner as 'ssd'.
rsi	Process to provide message passing over TCP/IP between DSI environments running on different machines.
rsi_lnk	Per link process created by rsi.
rsicmd	Configuration utility to configure individual RSI links.
tick	Protocol timer process to send periodic tick notification to the tim process that in turn handles protocol timers.
tim	Process to receive periodic tick notification from tick and handle protocol timers for all other processes.

4.2 Creating the System Configuration File (system.txt)

System configuration is handled by the gctload program that reads system configuration data from a file called system.txt. System initialization requires:

• First, that a pool of message buffers is created for subsequent inter-process communication.

• Second, that a message queue is created for each process that will run and that any message redirection for modules that are running remotely is initialized.

• Finally, that all processes can be started.

The gctload program handles this initialization sequence and creates the inter-process communication environment. The program reads input from the system.txt configuration file, carries out all system initialization and starts all processes.

The system.txt configuration file is a user-configurable file containing details of all the module identifiers known to the system, details of whether they are local modules or remote modules accessed by a local module (message redirection) and include the command line for all processes to be started by the gctload program.

The gctload program creates a message queue for each of the local module identifiers. The program subsequently expects a process to service its message queue, otherwise messages written to that queue will never be read causing eventual loss of system messages.

The gctload program initializes the message queue look-up table so that messages destined for modules that do not exist locally are redirected to a message queue for a module that does exists locally.

Having created the system environment, the gctload program proceeds to spawn all processes listed in the system.txt configuration file in the order listed.

Note: Prior to running the gctload program, the system.txt configuration file must be edited to reflect the requirements of your system.

4.2.1 System Configuration File Syntax

The system.txt configuration file is a text file used by the gctload program to configure the software environment. The file syntax permits the use of comments to improve the readability of the file. Comments are inserted into the file by using an asterisk (*). All characters on the line following the asterisk are ignored.

Numbers can be entered in either decimal or hexadecimal format. Hexadecimal numbers should be prefixed with 0x. For example, the value 18 can be entered in either of the following formats:

0x12 *(Hexadecimal)

18 *(Decimal)

The system configuration file can contain the following commands:

• LOCAL commands to allow the gctload program to generate message queues for modules running locally

• REDIRECT commands to cause messages generated for modules not running locally to be redirected via a module that is running locally.

• FORK_PROCESS commands advising the gctload program of any processes that need to be started locally

The full syntax of each command is listed in 7.1.1 System Configuration File (system.txt).

An example system.txt configuration file is shown in section 9.1.

4.2.2 Generating the system.txt Configuration File

This section describes the procedure for generating a system.txt configuration file and details operating system specific differences in behavior among the development packages.

First, the file must contain LOCAL declarations for modules that are to run on the host computer. For a board-based system, this must include the ssd module and the timer module. For a SIGTRAN system, ssd is not required:

```
LOCAL 0x20 * ssdh - Board interface task
LOCAL 0x00 * tim - Timer task
```

LOCAL declarations are also required for optional modules running on the host. Typically, this includes the s7_mgt protocol configuration utility and the user's own application module. It may also include any host-based protocol modules and the s7_log utility. For example:

```
LOCAL 0xcf * s7_mgt - Management/config task
LOCAL 0x2d * upe - Example user part task
LOCAL 0x3d * s7_log - Prints messages to screen/file
```

Additionally, a SIGTRAN system using M3UA requires LOCAL definitions for the SCTP and M3UA protocols. (the SCTPD module is only used when for systems that do not make use of the Native SCTP implementation).

```
LOCAL 0xd0 * SCTPD module (not required if SCTPN is being used)
LOCAL 0xd1 * SCTP or SCTPN module
LOCAL 0xd2 * M3UA module
```

Once all the LOCAL declarations are in place, REDIRECT commands should be added for modules that are running on the board so that messages destined for these modules are transported via ssd (module_id = 0x20) and the device driver to the board.

The following REDIRECT commands are always required for TDM-based systems:

```
REDIRECT 0x10 0x20 * CT Bus/Clocking control module
REDIRECT 0x8e 0x20 * On-board management module
```

For boards running MTP2 protocol layer a redirect command is required for all MTP2 module_id in use by the board. Usually this is just module_id=0x71 but for the SS7HD board there is a separate MTP2 module_id for each signaling processor. As follows:

```
REDIRECT 0x71 0x20 * MTP2 module_id (except SS7HD boards)

REDIRECT 0x81 0x20 * MTP2 module_id for SP 0 (SS7HD boards only)
REDIRECT 0x91 0x20 * MTP2 module_id for SP 1 (SS7HD boards only)
REDIRECT 0xe1 0x20 * MTP2 module_id for SP 2 (SS7HD boards only)
REDIRECT 0xf1 0x20 * MTP2 module_id for SP 3 (SS7HD boards only)
```

If ATM support is required (SS7MD boards only), then the following REDIRECT commands are also required:

```
REDIRECT 0x31 0x20 * ATM Module
REDIRECT 0x41 0x20 * Q.SAAL Module
```

For SIGTRAN systems using SCTPN the following REDIRECT is required:

```
REDIRECT 0xd0 0xd1 * required for SCTPN usage
```

For SIGTRAN systems using M3UA, the following REDIRECTS are required:

```
REDIRECT 0x22 0xd2 * redirect MTP3 to M3UA
REDIRECT 0xc2 0xd2 * mbm task now handled by M3UA
```

When using M3UA with multiple local AS, each additional local AS requires a redirect:

```
REDIRECT 0xd6 0xd2 * M3UA LAS2/NC1
REDIRECT 0xd7 0xd2 * M3UA LAS3/NC2
REDIRECT 0xd8 0xd2 * M3UA LAS4/NC3
```

In addition, REDIRECT commands are required for protocols running on the board. This typically includes MTP3 and one or more user parts. Examples of these commands are given below:

```
REDIRECT 0x22 0x20 * MTP3 module
REDIRECT 0x23 0x20 * ISUP module
REDIRECT 0x4a 0x20 * TUP module
REDIRECT 0x33 0x20 * SCCP module
REDIRECT 0x14 0x20 * TCAP module
```

Having provided that modules running on the board are accessible, it is then necessary to provide that status indications issued from the board successfully arrive at a module running on the host. If this does not happen, the system quickly runs out of available messages for inter-process communication.

Two module_id's (0xdf and 0xef) require redirection to a suitable process running on the host. Initially, these messages should be redirected to the s7_log utility that prints out a line for each message received. Ultimately, the user's own application will expect to receive these notifications.

```
REDIRECT 0xdf 0x3d * LIU/MTP2 status messages -> s7_log
REDIRECT 0xef 0x3d * Other indications -> s7_log
```

It is next necessary to include FORK_PROCESS commands for modules running on the host computer. All systems require tick and tim binaries to be run; therefore:

The mandatory FORK_PROCESS commands are:

```
FORK_PROCESS tim
FORK_PROCESS tick
```

Systems containing signaling boards require the appropriate ssdx process to be started:

```
FORK_PROCESS ssdx
```

For SIGTRAN implementations using the Native SCTP (SCTPN) and M3UA the following processes should be started:

```
FORK_PROCESS sctpn
FORK_PROCESS m3ua
```

For SIGTRAN implementations that use SCTP & SCTPD and M3UA the following processes should be started:

```
FORK_PROCESS sctpd
FORK_PROCESS sctp
FORK_PROCESS m3ua
```

Finally, FORK_PROCESS commands should be added for any other modules running on the host, such as, protocol modules, user applications or diagnostic utilities. For example:

```
FORK_PROCESS s7_mgt
FORK_PROCESS upe
FORK_PROCESS s7_log
```

4.3 Creating the Protocol Configuration File (config.txt)

The **s7_mgt** protocol configuration utility performs initialization of the protocol software modules. It reads the protocol configuration data from a text file, called config.txt, and provides a quick and flexible method of configuring the protocol modules without the need to write software for that purpose.

Alternatively, the protocol stack may be configured by sending the individual configuration messages documented in the per-module Programmer's Manuals for each protocol module. This approach can be of particular use when the application needs to make dynamic changes to protocol configuration without stopping the application program.

The command line syntax for s7_mgt is detailed in section 7.5. By default s7_mgt runs with module_id=0xcf and uses the filename "config.txt" for the Protocol Configuration File. Typically s7_mgt is started up from within system.txt using a FORK_PROCESS command. On completion of the single shot configuration sequence, s7_mgt will issue a notification message which can be used by an application as the indication that configuration is complete. The following example extract from system.txt starts s7_mgt so that it uses the default configuration file, and module_id and sends a notification to module_id 0xef:

```
FORK_PROCESS s7_mgt –i0xef
```

To assist diagnosis of issues, s7_mgt can be run using the -d option that generates additional diagnostic output.

The user should generate the config.txt file by reference to section 8 of this manual which details the syntax of all protocol configuration commands. In some cases it will also be necessary to refer to the Programmer's Manual for the specific protocols.

An example config.txt file is supplied as part of the DSI Development Package and this is repeated in section 9.2 of this manual. The example file shows typical usage of most commands although many of the commands are commented out by the use of '*' as the first character on the line.

4.4 Executing the Software

This section describes how to start the software running. It assumes that the software has already been installed and operating system adjustments have been made as detailed in section 3.

Ensure that device drivers for any boards have been installed and the system configuration file (system.txt) has been modified according to the system requirements.

Ensure that the location of any code file is consistent with the SS7_BOARD entry in config.txt (or the message based configuration parameters).

Ensure that the location of any host protocol binaries is consistent with the FORK_PROCESS entries in the system.txt file

If using s7_mgt, ensure that the protocol configuration file config.txt has been edited to provide the correct protocol configuration. Typically s7_mgt is started using a FORK_PROCESS command in system.txt).

To start the software running, change to the directory containing the gctload binary and run gctload, optionally specifying the system configuration file. To run the system in the background, enter:

```
gctload -csystem.txt &
```

For Windows users, gctload can be run in the background using:

```
start gctload -csystem.txt
```

The gctload program initializes the system environment and starts up other processes. The s7_mgt process configures the protocol modules. A banner confirms that the system is running.

To shutdown the DSI software environment, run gctload using the –x parameter (or if gctload was run in the foreground simply use CTRL-C). All modules that have been started by gctload are terminated automatically, for example:

```
gctload –x
```

The command line management utility dsictrl can be used to activate signaling links, for example:

```
dsictrl MTPL ACT 0-0
dsictrl MTPL ACT 0-1
```

Once gctload is running they status of the system can be observed by running a second instance of gctload using the –t1 parameter.

4.5 Developing a User Application

The development package, with the User Part Development Package, contains the files to allow the user to develop applications. These consist of makefile definitions, C header files (.h), and libraries.

A single definitions file is supplied (for each operating system) containing the definitions relating to the user's own development environment. This file is then included in the make files for all other processes. The user may need to modify this definitions file to ensure correct paths etc are set up.

Some simple example programs are supplied to illustrate techniques for interfacing to the protocol stack, although they are not intended to show a real application. Before starting to develop an application, you can familiarize yourself with the example programs and how they are built.

The example programs are contained in the User Part Development Package.

upe is a framework for a User Part module and contains a worked example of exchanging messages with the MTP3 module. It loops back any MTP-TRANSFER-INDICATIONS messages that it receives and reports other MTP indications to the user.

mtpsl is an example of how to send messages to MTP3 to activate and deactivate signaling links. It can be used as a command line tool for this purpose initially. It is intended that the user build the example code into the management application.

ctu is an example of how a user application can interface with telephony user parts, e.g., ISUP or TUP.

A makefile is included to allow users to build the application programs. To build the programs, change to the appropriate directory and enter (to build ctu):

nmake /f ctu.mak	(Windows)
make -f ctu.mak	(Linux)
make -f ctu.mak	(Solaris)

5 Message Reference

5.1 Message Format

5.1.1 MSG Message Structure

A message consists of a fixed header field and a variable length parameter field:

```
typedef struct msg
{
    HDR                 hdr;
    u16                 len;
    u32                 param[80]
} MSG;
```

hdr: The message header

len: This field indicates the number of bytes in the parameter area of the message. Some messages do not contain any data in the parameter area, in which case **len** is set to zero.

param: The parameter area of the message. The contents of this field are dependent on the message type. This field is normally accessed via a pointer obtained using the macro get_param(), which returns an unsigned character pointer to the param field.

The meaning of each field for a given message type is described in the individual application message specification. (See Programmer's Manual for each module).

The size of the **param** array is 320 bytes for a normal length message (ie as shoen above) or 4200 bytes for a long message.

Note: Users of systems where the structure 'MSG' is already defined for other purposes should use the alternative definition 'MSF'.

5.1.2 Header Fields

The application messages start with a common header, which is used to determine the message type, the source and destination module identities, and status information. This header structure is defined as follows:

```
typedef struct hdr
{
    u16 type;           /* type of message      */
    u16 id;             /* module instantiation */
    u8  src;            /* sending module ID    */
    u8  dst;            /* destination module ID*/
    u16 rsp_req;        /* response required    */
    u8  hclass;         /* generic MSG type     */
    u8  status;         /* returned status      */
    u32 err_info;       /* status information   */
#ifdef DSI_64BIT
    u32 next_ref;       /* reserved for internal use only */
#else
    struct hdr *next;   /* reserved for internal use only */
#endif
} HDR;
```

type: The type field is used to distinguish between different messages. It uniquely identifies the format of the remainder of the message and in particular the format of the message parameter area.

id: The id field allows modules, which handle multiple internal instances of a single entity (such as a circuit or signaling link) to distinguish the entity for which the message is destined.

src: The src field contains the module identity of the module that issued the message.

dst: The dst field contains the module identity of the module for which the message is destined.

rsp_req: The rsp_req field is used by the originator of a message to indicate whether or not it requires confirmation from the receiving module that the message has been received.

If the sending module requires confirmation, it sets a bit in the rsp_req field prior to sending the message. Which bit to set is determined by the value of the least significant nibble of the module's own module id (as written in the src field) For example, if the module id is 0x36 and message confirmation is required, the least significant nibble value of 0x6 indicates that the user would set bit 6 in the rsp_req field, so rsp_req would equal 0x0040.

If message confirmation is not required, then the rsp_req field should be set to zero.

The confirmation message takes the same format as the request message but uses a different **type** value. The **type** value for a confirmation message is derived from the **type** value in the request message by clearing bit 14.

hclass: This field is assigned by getm() and must not be modified.

status: This field is used for confirmation messages and indications to indicate the status associated with the message. A value of zero in a confirmation message usually indicates success.

err_info: This field is used in some confirmation and indication messages to supplement the status field and provide additional information

next: This field is reserved for internal use and must not be used.

5.1.3 Parameter Field

The parameter field for the standard MSG can contain 0 to 320 bytes of data. The data is stored in the parameter field in a host independent format. The contents and format of messages parameter field are defined in the various Programmer's Manuals. In order to support longer payload messages associated with some protocols, in addition to the standard MSG a 'long message' is supported, which offers up to 4,200 bytes in the parameter area. Long messages co-exist with the standard MSG and are only used as needed based on payload size.

When the message is allocated the length parameter specified in the getm() function call is used to determine whether to allocate a normal message (length <= 320) or a 'Long Message' (length > 320).

Internally two partitions are created, the first for standard 320 octet messages and the second for 4200 octet messages. The use of the second partition is optional and is enabled by the use the NUM_LMSGS command in the system.txt file which takes a single parameter representing the number of 'Long Messages' required in the system. This should be in the range 1 to 65,000.

5.2 Common Message Specifications

This section defines the inter process messages that are of generic use within the DSI software environment. Messages for individual protocol modules are specified in the appropriate protocol Programmer's Manual and control messages for specific boards are contained in the Programmer's Manual for the board.

This section defines the following message types:

- GEN_MSG_MOD_IDENT - Module Identification Request
- SYS_MSG_CONGESTION - Congestion Status Indication
- MGT_MSG_TRACE_EV - Trace Event Indication
- API_MSG_CNF_IND - Configuration Completion Status Indication

5.2.1 GEN_MSG_MOD_IDENT - Module Identification Request

Synopsis

Message issued to any module to read the module type and core revision number.

Format

MESSAGE HEADER	
Field Name	**Meaning**
type	GEN_MSG_MOD_IDENT (0x6111)
id	0
src	Sending module's ID
dst	Destination module_ID
rsp_req	Used to request a confirmation.
hclass	0
status	0
err_info	0
len	28

PARAMETER AREA		
Offset	**Size**	**Name**
0	2	Reserved
2	1	maj_rev
3	1	min_rev
4	24	text

Description

This message can be issued to any module to determine the module type and the core revision number of the internal software.

The confirmation message contains the major and minor revision numbers and a text string identifying the module.

Parameters

maj_rev

Major revision identifier for the object being queried.

min_rev

Minor revision identifier for the object being queried.

text

Null terminated string giving textual module identity (for example, "ss7.dc6").

5.2.2 SYS_MSG_CONGESTION - Congestion Status Indication

Synopsis

Message sent to the designated congestion handling module on change of system congestion state.

Format

MESSAGE HEADER	
Field Name	**Meaning**
type	SYS_MSG_CONGESTION (0x0001)
id	Partition_ id 0 - Indication relates to pool of standard length messages 1 - Indication relates to pool of 'long' messages.
src	0
dst	congestion-handling module
rsp_req	0
hclass	0
status	congestion_status
err_info	0
len	0

Description

When, as a result of allocating or releasing a message the system congestion status changes, this message is sent to the designated congestion handling module.

Parameters

congestion_status

The current congestion state of the DSI software environment. A value of zero indicates no congestion and non-zero values indicate various levels of congestion. Currently only 1 level of congestion is supported.

5.2.3 MGT_MSG_TRACE_EV - Trace Event Indication

Synopsis

Message issued by a module to trace a message sent to or received by the module.

Format

MESSAGE HEADER	
Field Name	**Meaning**
type	MGT_MSG_TRACE_EV (0x0003)
id	0
src	module_id of module generating the trace event
dst	management module id
rsp_req	0
hclass	0
status	0
err_info	Timestamp
len	18 + length of traced data

PARAMETER AREA		
Offset	**Size**	**Name**
0	1	src - hdr->src from traced message.
1	1	dst - hdr->dst from traced message.
2	2	id - hdr->id from traced message.
4	2	type - hdr->type from traced message.
6	2	status - hdr->status from traced message.
8	4	err_info – hdr->err_info from traced message.
12	4	par - pointer to hdr of message being traced.
16	2	data_length - number of bytes in data field.
18	0 to 280	data - Data taken from parameter area of traced message.

Description

For diagnostic purposes, each protocol module has the ability to trace messages sent, and received and certain management events. When a message is traced a copy of the message is embedded within the MGT_MSG_TRAVE_EV message and sent to the appropriate trace or management module. The user can dynamically change the configuration of which messages are traced using the per-module message to set the appropriate trace masks. Typically the trace messages are sent to the message queue of the s7_log utility which logs them to a rolling log file.

5.2.4 API_MSG_CNF_IND - Configuration Completion Status Indication

Synopsis

Message issued by s7_mgt protocol configuration utility on completion of initial configuration sequence.

Format

MESSAGE HEADER	
Field Name	**Meaning**
type	API_MSG_CNF_IND (0x0f09)
id	0
src	s7_mgt module_id (0xcf)
dst	Notification module (see below)
rsp_req	0
hclass	0
status	completion_status (see below)
err_info	Reserved for future use.
len	0

Description

This message is issued by the s7_mgt protocol configuration utility on completion of the configuration sequence and indicates either success (status=0) or an error condition that occurred during configuration. The message is only issued when s7_mgt is run with the –i command line option specifying the module ID of the Notification Module to which the message should be sent.

It is recommended that the user invoke this option, then wait for an API_MSG_CNF_IND message to provide that the application does not attempt to send messages until initial configuration is complete.

Parameters

completion_status

The result of initial configuration. The following table shows the possible values and their meanings.

Value	Meaning
0	Success
1	Error opening the config.txt protocol configuration file
2	Syntax or value error in the config.txt protocol configuration file
3	Error during configuration (invalid parameters)
4	Error during configuration (no response)

5.3 RSI Messages

RSI messages allow RSI links to be configured, activated and deactivated by the user. Once established they may also be used to gather the status and statistics on the link.

The following message types are defined:

- RSI_MSG_CONFIG – RSI Link Configuration Request
- RSI_MSG_UPLINK – RSI Link Activate Request
- RSI_MSG_DOWNLINK – RSI Link Deactivate Request
- RSI_MSG_LNK_STATUS – RSI Link Status Indication
- RSI_MSG_R_LNK_STATS – RSI Link Statistics Request
- RSI_MSG_READ_LINK – RSI Read Link Status

5.3.1 RSI_MSG_CONFIG – RSI Link Configuration Request

Synopsis

Message sent to the rsi module to configure an individual RSI link.

Format

MESSAGE HEADER	
Field Name	**Meaning**
type	RSI_MSG_CONFIG (0x7f80)
id	rsi_link_id
src	Sending module ID
dst	RSI module ID (0xb0)
rsp_req	Used to request a confirmation
hclass	0
status	0
err_info	0
len	130

PARAMETER AREA		
Offset	**Size**	**Name**
0	1	reserved – must be set to zero
1	1	conc_id
2	2	flags
4	2	local_port
6	2	remote_port
8	20	reserved – must be set to zero
28	20	remote_addr
48	2	reserved – must be set to zero
50	80	peer_addr

Description

The RSI_MSG_CONFIG message is used to configure an RSI link. For correct operation one end of the link must be configured as a client and the other as a server. The link is initialized in the out of service (inactive) state and can subsequently be brought into service using the RSI_MSG_UPLINK message.

Network addresses can be specified as DNS hostname, IPv4 or IPv6 addresses. All addresses are specified as null terminated ASCII strings.

For example:

IPv4 address:	123.124.125.126
IPv6 link local address via eth0:	fe80::20d:60ff:feb7:d751%eth0
IPv6 global address:	19a9:8cf0:0:20d:60ff:feb7:d751
DNS address:	dpkbuild.lab.yourcompany.com

Parameters

rsi_link_id

The local logical RSI link identifier in the range 0 to one less than the number of links supported.

conc_id

The concerned module_id to which RSI link status indications will be sent.

flags

A 16-bit value specifying additional run-time configuration options.

flags	Description
Bit 0	Client / Server setting. This bit should be set to 0 for the Client end of the link and set to 1 for the Server end or the link.
Bit 1	Reserved for future use and should be set to zero.
Bit 2	This bit should be set to 1 on system that support ‘long’ messages.
Bits 3 to 15	All other bits are reserved for future use and should be set to zero.

local_port

The local port number for a server link. (This should be set to zero for client links).

remote_port

The remote port number for a client link. (This should be set to zero for server links).

remote_addr

Retained for backwards compatibility only.

peer_address

Holds either the peer’s Network Address, or an IPv4 or IPv6 address as null terminated ASCII string.

5.3.2 RSI_MSG_UPLINK – RSI Link Activate Request

Synopsis

Message sent to the RSI module to activate an individual RSI link.

Format

MESSAGE HEADER	
Field Name	**Meaning**
type	RSI_MSG_UPLINK (0x7f81)
id	rsi_link_id
src	Sending module ID
dst	RSI module ID (0xb0)
rsp_req	Used to request a confirmation
hclass	0
status	0
err_info	0
len	0

Description

The RSI_MSG_UPLINK message is sent to RSI to activate a previously configured rsi link. The rsi process attempts to establish the link on receipt of this message. In the event that the link subsequently fails, the RSI module will automatically attempt to restore it.

5.3.3 RSI_MSG_DOWNLINK – RSI Link Deactivate Request

Synopsis

Message sent to the RSI module to deactivate an individual RSI link.

Format

MESSAGE HEADER	
Field Name	**Meaning**
type	RSI_MSG_DOWNLINK (0x7f82)
id	rsi_link_id
src	Sending module ID
dst	RSI module ID (0xb0)
rsp_req	Used to request a confirmation
hclass	0
status	0
err_info	0
len	0

Description

The RSI_MSG_DOWNLINK message is sent to RSI to take an RSI link out of service.

5.3.4 RSI_MSG_LNK_STATUS – RSI Link Status Indication

Synopsis

Message issued by RSI to indicate changes in status of the RSI link.

Format

MESSAGE HEADER	
Field Name	**Meaning**
type	RSI_MSG_LNK_STATUS (0x0f83)
id	link_id
src	RSI module ID (0xb0)
dst	Concerned ID
rsp_req	0
hclass	0
status	Link State
err_info	0
len	0

Description

The RSI_MSG_LNK_STATUS message is issued by RSI to the concerned module (as configured at RSI link configuration) whenever the RSI link goes in service or out of service.

Parameters

Link State

The status of the RSI link as follows:

Value	Link state
1	Link established (In Service)
2	Link failed (Out of Service)

5.3.5 RSI_MSG_R_LNK_STATS – RSI Link Statistics Request

Synopsis

Message sent to RSI to read (and optionally reset) statistics for an individual RSI link.

Format

MESSAGE HEADER	
Field Name	**Meaning**
type	RSI_MSG_R_LNK_STATS (0x6f87)
id	rsi_link_id
src	Sending module ID
dst	RSI module ID (0xb0)
rsp_req	Used to request a confirmation
hclass	0
status	Set to 0 to read statistics Set to 1 to read and reset statistics
err_info	0
len	36

PARAMETER AREA		
Offset	**Size**	**Name**
0	1	version
1	3	spare
4	4	period
8	4	tx_msgs
12	4	rx_msgs
16	4	tx_kbytes
20	4	rx_kbytes
24	4	oos_duration
28	4	oos_count
32	4	tx_discards

Description

The RSI_MSG_R_LNK_STATS message is used to read back statistics from the rsi link. The sending module should set to 'version' parameter to zero and should ensure that a confirmation is requested. The RSI module will populate the remaining parameters in the parameter area in the confirmation message. The statistics can optionally be reset by setting the 'status' to 1.

Parameters

period

The time period over which the statistics have been gathered (in multiples of 100ms).

tx_msgs

Number of messages transmitted over the link within the measurement period.

rx_msgs

Number of messages received over the link within the measurement period.

tx_kbytes

Number of octets transmitted in messages over the link within the measurement period. Excludes the message header.

rx_kbytes

Number of octets received in messages over the link within the measurement period. Excludes the message header.

oos_duration

The total amount time the link was out of service during the measurement period (in multiples of 100ms).

oos_count

The number of times the link went out of service during the measurement period.

tx_discards

The number of messages due to be transmitted on the link that were discarded during the measurement period.

5.3.6 RSI_MSG_READ_LINK – RSI Read Link Status

Synopsis

Message used to read the current status and parameters of an RSI link.

Message Format:

MESSAGE HEADER	
Field Name	**Meaning**
type	RSI_MSG_READ_LINK (0x6f84)
id	rsi_link_id
src	Sending module ID
dst	RSI module ID (0xb0)
rsp_req	Used to request a confirmation
hclass	0
status	0
err_info	0
len	130

PARAMETER AREA		
Offset	**Size**	**Name**
0	4	reserved – must be set to zero
4	2	lport
6	4	reserved – must be set to zero
10	2	fport
12	2	tcpstate
14	18	host_addr
32	18	peer_addr

50	80	peer_name

Parameters

lport
Local port

fport
Peer port

tcpstate
State of the underlying TCP connection

host_addr
Holds the local IPv4 or IPv6 address of the connection. The message supports both IPv4 and IPv6 addresses.

For an IPv4 connection, the first byte is set to 1 followed by a 32 bit IPv4 address.

For an IPv6 connection, the first byte is set to 2 followed by a 128 bit IPv6 address and in the case of a link local address the scope (or 0xff for a non link local address).

peer_addr
Holds the remote IPv4 or IPv6 address of the connection. The message supports both IPv4 and IPv6 addresses.

For an IPv4 connection, the first byte is set to 1 followed by a 32 bit IPv4 address.

For an IPv6 connection, the first byte is set to 2 followed by a 128 bit IPv6 address and in the case of a link local address the scope (or 0xff for a non link local address).

peer_name
For client end, this parameter is the name used at configuration time. For server end this parameter is set to a null string.

6 Library Functions

6.1 Inter-Process Communications Functions

6.1.1 GCT_send

Synopsis

Function to send a message to the specified module_id.

Prototype

```
int GCT_send(unsigned int module_id, HDR *h);
```

Return Value

Returns zero on success, non-zero otherwise.

Parameters

module_id - The destination module id. This will usually be the same as the value contained in the **hdr.dst** field of the message.

h - A pointer to the HDR structure at the start of the MSG to be sent. This parameter should always point to a buffer allocated using getm().

Description

This function uses **module_id** to determine which message queue the message should be sent to and sends the message. A success return value implies that the message has been sent to the message queue belonging to the destination process.

If the call is successful, the calling program no longer owns the message and must no longer access it. If the function does not return success, then the calling program is responsible for the release of the message back to the system using relm().

6.1.2 GCT_receive

Synopsis

Function to wait until the next message for **module_id** is available and return a pointer to the message.

Prototype

```
HDR *GCT_receive(unsigned int module_id);
```

Return Value

A pointer to the received message on success or zero on failure (in which case the user should retry the call).

Parameters

module_id - The module's own module id.

Description

This function uses **module_id** to determine from which message queue to receive. If the message queue contains a message, then a pointer to the first message is returned. Otherwise, the function suspends the calling task until a message is available.

After processing, the message returned by the GCT_receive function must either be sent back to the sending module (as a confirmation message), released back to the system using relm() or forwarded to a third module.

The only difference between GCT_receive and GCT_grab is whether to block or not when no messages are available.

6.1.3 GCT_grab

Synopsis

Function to determine whether there is a message ready for **module_id** and return a pointer to the message. If no message is ready, then the function returns immediately.

Prototype

```
HDR *GCT_grab(unsigned int module_id);
```

Return Value

A pointer to the received message on success or zero if there are no messages waiting.

Parameters

module_id - The module's own module id.

Description

This function uses **module_id** to determine from which message queue to receive. If the message queue contains any messages, then a pointer to the first message is returned. Otherwise the function immediately returns zero.

After processing, the message returned by the GCT_grab function must either be sent back to the sending module (as a confirmation message) or released back to the system using relm() or forwarded to a third module.

The only difference between GCT_receive and GCT_grab is whether to block or not when no messages are available.

6.1.4 GCT_set_instance

Synopsis

Function to write the module **instance** into the message pointed to by **h.**

Prototype

```
int GCT_set_instance(unsigned int instance, HDR *h);
```

Return Value

Returns zero on success, non-zero otherwise (currently no failure conditions are defined).

Parameters

instance - The destination module instance.

h - A pointer to the HDR structure at the start of the MSG.

Description

Writes the destination module instance into the message. This function should be called prior to calling GCT_send by the module sending the message.

The destination module instance is used when messages are sent from one processor to another processor. It determines the destination processor to which the message is sent.

Examples of the use of this function are as follows:

a) When sending messages to one of several boards. In this case, the module instance is the board_id.

b) When sending messages to one or other Dialogic® DSI Signaling Interface Unit (SIU) from an SIU pair. In this case, the module instance is 0 (SIUA) or 1 (SIUB).

6.1.5 GCT_get_instance

Synopsis

Function to recover the module instance from the message pointed to by **h.**

Prototype

```
unsigned int GCT_get_instance(HDR *h);
```

Return Value

Returns the module instance read from the message.

Parameters

h - A pointer to the HDR structure at the start of the MSG.

Description

Recovers the source module instance from a received message. This function should be called after return from GCT_receive or GCT_grab.

The source module instance is used when messages are received from a number of processors by the local module. It identifies the source processor at which the message originated.

Examples of the use of this function are as follows:

a) When receiving messages from one of several boards. In this case, the module instance is the board_id.

b) When receiving messages from one or other Signaling Interface Unit (SIU) in an SIU pair. In this case, the module instance is 0 (SIUA) or 1 (SIUB).

6.1.6 getm

Synopsis

Function to allocate an MSG and initialize given fields in the message header.

Prototype

```
MSG *getm(unsigned short type, unsigned short id,
          unsigned short rsp_req, unsigned short len);
```

Return Value

A pointer to the allocated message, or zero if no message available.

Parameters

type - The message type, this is written to the **hdr.type** field of the message before the function returns.

id - The id value; this is written to the **hdr.id** field of the message before the function returns.

rsp_req - The rsp_req value; this is written to the **hdr.rsp_req** field of the message before the function returns (Refer to 5.1.2 Header Fields).

len - The number of bytes that the user wishes to place in the parameter area of the message. This is written to the **len** field of the message before the function returns. This field is used to determine whether to allocate a standard message or a long message. If len is less than or equal to 320, then a standard message is allocated. If len is between 321 and 4200 inclusive, then a long message is allocated.

Description

This function allocates a message buffer from the buffer pool and initializes the type, id, rsp_req and len fields of the message to the specified values.

The function is used to allocate a message for subsequent inter-process communication, where it will be sent to the destination process. On return from the function, it is the calling functions responsibility to initialize the **hdr.src** and **hdr.dst** fields and the parameter area of the message prior to calling GCT_send().

6.1.7 relm

Synopsis

Function to release a message that has previously been allocated by getm(), back to the system.

Prototype

```
int relm(HDR *h);
```

Return Value

Zero on success; non-zero otherwise.

Parameters

h - A pointer to the HDR structure at the start of the MSG.

Description

Returns a message buffer allocated by getm() to the system buffer pool.

Each message allocated must be returned once (and only once) to the system. It does not need to be returned by the same process that allocated it.

6.1.8 GCT_link

Synopsis

Function optionally used to attach an application to the DSI software environment, and detect the existence of the environment.

Prototype

```
int GCT_link(void);
```

Return Value

Returns zero on success.

Non zero is returned on failure, indicating that gctload is not running (GCT environment is not available).

Description

This optional function is called by an application that wishes to confirm the existence of the DSI software environment in advance of using it. Refer to section 2.8 for further details.

Typically this function is not needed. The first call by an application to GCT_grab, GCT_receive or getm will automatically attach to the DSI environment.

6.1.9 GCT_unlink

Synopsis

Function optionally used to force an application to detach from the DSI software environment.

Prototype

```
int GCT_unlink(void);
```

Return Value

Always returns zero.

Description

This optional function is called by an application that wishes to forcibly unlink from the DSI software environment (for example to allow the DSI software environment ot be restarted without needing to restart the application). Refer to section 2.8 for further details.

Prior to calling GCT_unlink() the application must ensure that all messages have been released back to the environment.

Typically this function is not needed. When a module terminates it automatically unlinks from the DSI software environment.

6.1.10 GCT_partition_congestion

Synopsis

Function used to determine the congestion status of the DSI software environment.

Prototype

```
int GCT_partition_congestion (int partition_id);
```

Parameters

The **partition_id** identifies the particular message pool. It should be set to 0 for the pool of standard MSGs and 1 for the pool of 'Long' messages.

Return Value:

The return value is set to the current congestion state of the DSI software environment. A value of zero indicates no congestion and non-zero values indicate various levels of congestion. Currently only 1 level of congestion is supported.

Description

The congestion status is determined by the number of messages currently allocated as a percentage of the total number of messages within the message pool. When a system is under heavy load there may be insufficient CPU power to process the incoming messages as fast as they are received so the number of messages queued within the environment starts to increase. Usually this is a transient condition and the load over time balances out and the congestion clears. A second cause of congestion is when messages are sent to a message queue which is not being serviced by an active process. A further cause of congestion is when modules do not release messages back into the environment. If the number of messages currently allocated increases above a threshold the congestion status will be set to 1. This function allows an application to determine the current congestion status of the system.

See also gctload –t1 which provides similar information from the command line.

6.1.11 confirm_msg

Synopsis

Function to confirm a message once it has been handled.

Prototype

```
int confirm_msg(MSG *message);
```

Parameter

The **message** is a pointer to the message to be confirmed.

Return Value

The function always returns 0.

Description

This function is called when a module has finished processing a message. If the sending layer's response required bit is set, then the message is converted to a confirmation message and sent back using GCT_send() to the sending module. If no confirmation was requested then the message is released back to the software environment using the relm().

A confirmation message is generated by swapping the hdr.src and hdr.dst fields, clearing bit 14 in the hdr.type field and clearing the sending layer's bit in the hdr.rsp_req field.

The confirm_msg() function is the preferred way for an application to release a message back to the system once it has finished processing the content. It takes care of inspecting the **rsp_req** field to determine whether a confirmation is required, and it adjusts the **type** field if necessary and calls either **relm** or **GCT_send** simplifying the application code and reducing the risk of errors.

6.2 General Library Functions

This section details other useful functions that are built into the gctlib library.

6.2.1 rpackbytes

Synopsis

Function to pack bytes into machine independent format.

Prototype

```
void rpackbytes(unsigned char *dest, int dest_byte_offset,
                unsigned long value, int bytecount);
```

Return Value

None.

Parameters

dest - pointer to the destination buffer

dest_byte_offset - offset from the start of the destination buffer to store data

value - the value to be put into the buffer

bytecount - the number of significant bytes to take from value.

Description

Packs the requested number of bytes into a buffer in a machine-independent manner for sending to another module, regardless of byte ordering on either processor. Typically this function is used to populate configuration messages with the appropriate data.

Example

```
rpackbytes(dest, 10, value, 2);
```

This call will use the least significant 2 bytes of the value and store the resulting data starting at location dest + 10. The least significant byte of value will be written to dest + 11 and the next significant byte to dest + 10.

6.2.2 runpackbytes

Synopsis

Function to extract bytes from machine-independent format.

Prototype

```
unsigned long runpackbytes(unsigned char *src, int src_byte_offset,
                           int bytecount);
```

Return Value

The numeric value unpacked from the buffer. The user should cast this to the required type (u8, u16, u32 etc).

Parameters

src - pointer to the source buffer

src_byte_offset - offset from the start of the message buffer to retrieve data

bytecount - the number of bytes to take from the message

Description

Unpacks the requested number of bytes from a buffer, regardless of byte order on the processor.

Example:

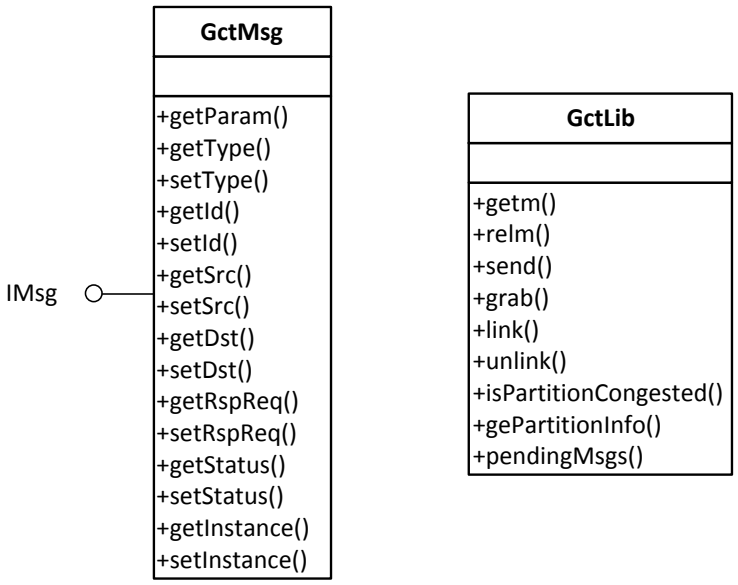
```
result = (u16)runpackbytes(src, 10, 2);
```

This call will retrieve the least two significant bytes from the buffer src and return the value as a u16. The u16 will be formed by src + 11 as the least significant byte and src + 10 as the most significant byte.

6.3 Java Inter-Process Communications

As an alternative to the C library functions discussed in section 6.1 the DSI Development Package includes a set of Java classes that provide equivalent functionally for use in a Java environment. The classes are included in the `gctApi.jar` within the JAVA sub-directory of the Development Package.

There are two key classes in this package.



GctLib

This class controls the access into the Message-Passing environment. It provides methods equivalent to the functions listed in section 6.1 Inter-Process Communications Functions.

Allocating a message:

```
GctMsg txMsg = GctLib.getm(MessageLength);
```

Sending a message:

```
GctLib.send(DestModId,txMsg);
```

Receiving a message:

```
GctMsg rxMsg = GctLib.receive(LocalModuleId);
```

GctMsg

This class provides a wrapper around a C message structure to allow it to be used in an alternative language.

The full list of classes and methods for the package are listed in Appendix C - GCTLIB Javadoc.

Example Code to display a message:

```
try {
     System.out.print(prefix
             + "M-t" + String.format("%04x", gctmsg.getType())
             + "-i" + String.format("%04x", gctmsg.getId())
             + "-f" + String.format("%02x", gctmsg.getSrc())
             + "-d" + String.format("%02x", gctmsg.getDst())
             + "-s" + String.format("%02x", gctmsg.getStatus()));


     ByteBuffer buf = gctmsg.getParam();

     if (buf.hasRemaining()) {
         System.out.print("-p");

         while (buf.hasRemaining()) {
             System.out.print(String.format("%02x", BBUtil.getU8(buf)));
         }
     }

     System.out.print("\n");

 } catch (GctException gctEx) {
     System.out.println("Problem with message: " + gctEx.getMessage());
 }
```

This example shows the message body being read via the use of the ByteBuffer class. The ByteBuffer can also be used to manipulate the message body to add parameters.

7 Host Utilities

7.1 gctload

Description

The DSI software environment is created and maintained using the gctload utility. All DSI implementations use gctload.

gctload reads user supplied system configuration parameters from the System Configuration File. The filename of this file by default is "system.txt" although an alternative filename can be used if the –c option is specified. Within this manual it is often simply referred to as system.txt.

The system.txt file details the number and type of all messages to allocate, it lists all the module identifiers known to the system (including details of whether they are local modules or remote modules accessed through message redirection) and lists the command lines for all processes to be started by gctload. The file also contains configuration parameters for congestion management and a number of optional commands.

gctload uses the **NUM_MSGS** and **NUM_LMSGS** commands to build pools of message buffers for subsequent use by the getm() and relm() functions.

gctload creates a message queue for each of the **LOCAL** module identifiers. It subsequently expects a process to service each message queue; otherwise, messages will be written to message queues and never read causing a loss of messages.

gctload uses the **REDIRECT** commands to initialize the message queue look-up table so that messages destined for remote modules can be re-directed via the appropriate LOCAL module.

gctload uses the **CONG_MSG** command to initialize congestion reporting parameters and thresholds.

Having created the system environment, gctload uses the **FORK_PROCESS** commands to spawn all processes listed in the system configuration file. It then remains dormant until it receives a signal from the user (using gctload –x) to shutdown the system.

To shut down the system, it terminates any processes that it created and releases all system resources back to the operating system.

The gctload utility can also be run a second time with one of the options (-t1, -t2, -t3 or –t4) in order to retrieve status information relating to the DSI software environment.

Syntax

```
gctload -v
gctload [-c<system config file> -d ]
gctload -x
gctload –t1
gctload –t2
gctload –t3
gctload –t4
```

Parameters

-v

Show version information.

-c<system config file>

The system configuration file contains full details of message queues, module_ids and local processes. This parameter is the filename of that file which by default is "system.txt". The format of the system configuration file is detailed fully in section 7.1.1, System Configuration File (system.txt) on page 81.

-d

Enables additional diagnostic output during creation of the DSI software environment.

-x

This option is used to terminate an existing gctload session. It ensures that the environment is shutdown in a controlled manner and that all processes forked within the system.txt file are also shutdown. This is the preferred way to shutdown the DSI software environment.

-t1
-t1r

The –t1 option is used to obtain a report on the current status of the DSI software environment. gctload should already have been run (without the –t1 option) so the DSI software environment is operational, running gctload a second time using the –t1 option will interrogate the current status.

The status output shows the key configuration parameters and current status values and is intended as a diagnostic tool to monitor the health of the system. The example below shows typical usage.

The **–t1r** form of the option additionally resets certain measurements (`'Max alloc since reset'` and `'Cong count since reset') and the` associated time stamps.

```
gctload –t1

GCTLOAD System Status:      2012-03-06 16:52:46.112

  System restart time:      2011-03-06 16:52:46.

  Congestion module Id:     0x21
  GCTLIB library:           V1.44
  Internal system error:    0
  GCTLIB Atomic:            Enabled
  Timed licenses in use:    No
```

```
  Partition[0]
    Parameter size:         320
    MSGs in partition:      5000
    MSGs allocated :        0
    MSGs free:              5000
    Maximum MSGs allocated: 13
    Max alloc since reset:  12
    Time of last max        2012-03-06 16:52:46.112
    Out of MSG count:       10
    Congestion onset:       2500
    Congestion abate:       500
    Congestion status:      0
    Congestion count:       2
    Cong count since reset: 1
    Last congestion onset:  2012-03-06 16:52:46.112
  Partition[1]
    Parameter size:         4200
    MSGs in partition:      10
    MSGs allocated :        0
    MSGs free:              10
    Maximum MSGs allocated: 8
    Max alloc since reset:  7
    Time of last max        2012-03-15 13:02:23.178
    Out of MSG count:       10
    Congestion onset:       5
    Congestion abate:       1
    Congestion status:      0
    Congestion count:       2
    Cong count since reset: 1
    Last congestion onset:  2012-03-15 13:02:23.178
```

-t2

The –t2 option displays a list of all the currently allocated messages to the console. These messages are shown in the same format as described for the s7_log and s7_play programs. Typically the –t2 option is used after having identified (using the –t3 option) that unexpected messages are queued within the environment in order to understand which message types are involved. Example output is shown below:

```
gctload –t2

GCTLOAD Allocated MSGs:     2012-03-06 16:52:46.112

M-I0000-t7680-i0000-fcf-d20-s00
M-I0001-t7681-i0000-fcf-d20-s00
M-I0000-t7203-i0000-fcf-d71-s00-p(9)112233445566778899
```

-t3

The –t3 option displays the current message queue status for all local message queues. This includes the number of messages currently queued and the process id (pid) of the last process to read from the message queue. To use the option the user should run a second instance of gctload using the –t3 option.

Under normal operation, the message queues for destination tasks should either be empty or contain a small number of messages. If this is not the case, this may be due to one of the following reasons:

- No active task has been set to read messages for the listed destination
- The destination task may have stopped reading from its message queue or may have stopped running.
- There may be a missing REDIRECT statement in the hosts' system.txt file to redirect messages from the listed destination to a running task.

```
gctload -t3

GCTLOAD Message Queue Status: 2012-03-06 16:52:46.112

LOCAL=0x00, MSGs queued=6, last read by pid=1167
LOCAL=0xef, MSGs queued=0, last read by pid=1182
```

-t4

The –t4 option displays the license status of all active DSI host software licenses and, in the case of time limited licenses, shows the expiry date for the license.

```
gctload –t4

GCTLOAD License Status: 2012-03-06 16:52:46.112

pid        Token         Expires
123456     MTP3_LNX      29-Feb-2012
120        MTP2MD256_LNX -
```

7.1.1 System Configuration File (system.txt)

The system configuration file (system.txt) is used by **gctload** to configure the DSI software environment. The system.txt file is tailored by the user to include the appropriate set of protocols and utilities. This section details the format of the system.txt file and defines the commands and parameters that can be used in the file.

The file syntax permits the use of comments to improve readability. Comments are inserted into the file by using an asterisk (*); all characters on the line after the asterisk are ignored.

Numbers can be entered in either decimal or hexadecimal format. Hexadecimal numbers should be prefixed with 0x. For example the value eighteen can be entered in either of the following formats:

```
0x12       * (Hexadecimal)
18         * (Decimal)
```

The System Configuration File commands allow local modules to be declared (each local module requires a message queue), messages for remote modules to be redirected via the appropriate interface module (eg ssd, rsi etc) and command lines for processes to be started up to be listed. The syntax of each command is listed in the following sections.

7.1.2 NUM_MSGS / NUM_LMSGS Commands

Synopsis

Configure the number of messages in the standard and large message pools.

Syntax

```
NUM_MSGS <num msgs>
NUM_LMSGS <num_lmsgs>
```

Example

```
NUM_MSGS 5000
NUM_LMSGS 200
```

Description

This command configures the number of messages globally allocated for use within the DSI software environment. All systems need to have a pool of 'standard' messages configured using the NUM_MSGS command. Optionally, systems may also have a pool of 'long' messages configured using the NUM_LMSGS command. Long messages are typically required for the use of transaction-based protocols where SCCP is performing segmentation and reassembly.

7.1.3 CONG_MSG Command

Synopsis

Configures the congestion parameters for the DSI software environment.

Syntax

```
CONG_MSG <module Id> <onset threshold> <abatement threshold>
```

Example

```
CONG_MSG 0x21 50 10
```

Description

This command configures the behavior of the congestion reporting system, as detailed in section 2.9. The following parameters can be configured using this message.

The congestion **module Id** specifies the module to which a congestion notification message is to be sent in the event of system congestion onset or abatement.

The congestion **onset threshold** specifies the percentage of the total number of available messages that must be allocated before the system will start congestion procedures.

The congestion **abatement threshold** specifies the percentage of the total number of messages that must be available before the system will terminate congestion procedures.

7.1.4 LOCAL Command

Synopsis

Command to create a message queue for a given module identifier that will be serviced by a local module.

Syntax

```
LOCAL              <module_id>
```

Example

```
LOCAL      0x20   * Create message queue for module_id 0x20
```

Description

This command causes gctload to create a message queue and associate the queue with the given module_id.

These commands should appear prior to any redirect commands. One entry should appear for each local module that will run in the system. The module identifier, <module_id>, must be in the range 0x00 to 0xfe and must not have already been declared. Usually, the module_id is entered in hexadecimal format.

7.1.5 REDIRECT Command

Synopsis

Command to cause messages for a given module identifier to be redirected to an alternative message queue.

Syntax

```
REDIRECT <new_module_id> <existing_module_id>
```

Example

```
REDIRECT  0x22  0x20  * Redirect messages for 0x22 to module 0x20
```

Description

This command causes messages destined to <new_module_id> to be redirected to <existing_module_id>. The <existing_module_id> must have already been declared as a local module.

Messages for many module identifiers may be re-directed to a single module. A separate command line should be used in each case.

Typical use for this command is to redirect messages intended for processes that are running on a remote board via a local process which is responsible for transferring the message to the remote board.

7.1.6 DEFAULT_MODULE Command

Synopsis

Command to cause messages for any module identifier not explicitly defined to be redirected to an alternative message queue.

Syntax

```
DEFAULT_MODULE <default_module_id>
```

Example

```
DEFAULT_MODULE  0xef     * Redirect messages by default to module 0xef
```

Description

This command saves using several REDIRECT commands and allows messages for any unspecified module_id to be redirected to a single default module_id. It is good practice to always include the DEFAULT_MODULE command to ensure that all module identifiers are serviceable.

7.1.7 FORK_PROCESS Command

Synopsis

Command to start up processes within the DSI software environment.

Syntax

```
FORK_PROCESS  <process_path_name> { <parameters> }
```

Example

```
FORK_PROCESS  /mydir/BIN/myproc                * Startup my process
```

Description

This command causes the specified process to be spawned once the system environment has been created. Command line parameters for the process can also be specified, although there may be some limitations to the symbols that are permitted.

The maximum number of FORK_PROCESS commands supported in a system.txt configuration file is 64. A process does not have to be spawned in the configuration file, provided it is run after **gctload** and its module identifier has been declared as local. An advantage of using the configuration file is that the processes spawned by gctload automatically get shutdown when using gctload –x to shutdown the DSI software environment.

7.1.8 Example system.txt File

The following example of a generic system.txt file creates and assigns a message queue to module_ids 0x10, 0x20 and 0x30, and redirects messages for module_id 0x40 to module_id 0x20. It then starts up processes to service the message queues for the local modules. An example real-world system.txt file is shown in section 9.1.

```
*     Example System Configuration File - system.txt
*
*
NUM_MSGS    5000
*
CONG_MSG    0x21 50 10
*
LOCAL       0x10          * Process_A's Module ID
LOCAL       0x20          * Process_B's Module ID
LOCAL       0x30          * Process_C's Module ID
*
REDIRECT    0x40   0x20   * Redirect msgs for Module ID 0x40 to Process_B
*
DEFAULT_MODULE     0x30   * Send messages for any unspecified Module ID to Process_C
*
NUM_MSGS    5000          * Number of standard size messages in the environment
NUM_LMSGS   200           * Number of 'long' messages (used for certain TCAP based
applications)
*
FORK_PROCESS  Process_A
FORK_PROCESS  Process_B
FORK_PROCESS  Process_C
```

7.2 s7_log

Synopsis

The s7_log utility services a specified message queue, receiving all messages and generates text based output either to the screen or to a log file. Maintenance and status events are interpreted as text; other messages are displayed in raw hexadecimal format. All entries in the log file are timestamped with date and time.

The utility is able to generate rolling, size limited log files and is suitable for real-time logging of messages to disk. Typically one or more instances of s7_log will be present in a system. For example one instance might log management events and status indications whilst other instances could be used to log measurements or to log traces protocol messages.

Syntax

```
s7_log [ -v –m<module_id> -o<options> -f<filename> -n<num_log_files>
         -s<logfile_size> -p<PCAP_filename> -t<timestamp> -q –r –w -x]
```

Parameters

-v

Show version information.

-m<module_id>

The module identifier assigned to the s7_log process. If not specified, s7_log will use a module ID of 0xef. The module ID assigned to s7_log must have a corresponding LOCAL entry in the system.txt file and must not be in use by other processes.

-o<options>

A 16 bit value that specifies the type of message reporting that will occur. If not specified, a value of 0xaf0d is used. Each bit set to 1 enables reporting of a particular message group or parameter field as described in the following table:

Bit	Function
0	Enable text interpretation of all recognized messages.
1	Display ALL received messages (including those interpreted as text) as hexadecimal.
2	Decode and display Management trace messages.
3	Decode and display Management Trace Event ‘time stamp’ field.
4	Decode message header src and dst fields as text if recognized.
5	Enables the decoding of timestamps included in API_MSG_RX_INDT messages received from DSI SS7 Boards. Setting bit 5 to 1 specifies the timestamp values taken from the internal board clock should be displayed in short form (time only). The timestamp information is displayed after the “BRD:“ label in the log. Note: This timestamp is different and more precise than the timestamp derived from the host clock, enabling usage of the -t[t|d] option as described below.
6	As for bit 5, it enables the decoding of timestamps included in API_MSG_RX_INDT messages received from DSI SS7 Boards. Bit 6 differs from bit 5 by displaying the timestamp values taken from the internal board clock in long format (date and time). Setting bit 6 to 1 overrides the value of bit 5 and always results in the display of timestamps in the long format. If both bits 5 and 6 are set to 0, the timestamp is not displayed.
7	Not used. Must be set to zero.
8	Display message type field.

Bit	Function
9	Display message id field.
10	Display message src field.
11	Display message dst field.
12	Display message rsp_req field.
13	Display message status field.
14	Display message err_info field.
15	Display message parameter field.

-f<filename>

Optionally specifies a text file to which the output from s7_log will be written. s7_log will create a backup of the existing log file, if one exists, with the filename <logfile_name>.old. When operating with rolling log files using the –s and –n options s7_log will not create the backup file.

-n<num_log_files>

Optionally allows multiple log files to be created in a rolling log format to prevent filling the hard drive. This parameter should be set to a value between 2 and 99 to control how many log files are created. The filenames of the log files will be in the following form. Each time the latest file is full, each file is renamed.

`log.txt` (most recent file),
`log.txt.1` (second most recent file)
…
`log.txt.[n-1]` (oldest file)

-s<logfile_size>

Use in conjunction with the –n option to specify the maximum file size (in kbytes) for a rolling log file. The valid range is from 1 to 100,000 representing log file sizes from 1kbytes to 100,000kbytes.

-p<PCAP_filename>

The –p option causes a PCAP formatted log file with the given filename to be created. s7_log will log the following message types in the PCAP format file: API_MSG_RX_IND, API_MSG_RX_INDT and API_MSG_TX_REQ.

When running in terminated mode, the user needs to activate tracing of MTP3 TX_REQ in the output event trace mask and RX_IND in the input trace mask and these trace messages will be logged into the PCAP format log file.

-tt, -td, -tp

The –tt, –td and -tp options cause each entry created by s7_log to contain a timestamp.

Option	Output	Format
-tt	Displays time only	hh:mm:ss:ddd
-td	Displays date and time	YYYY-MM-DD hh:mm:ss.ddd
-tp	Displays date and higher precision timestamps to micro-second granularity	YYYY-MM-DD hh:mm:ss.dddddd

-q

Optional quiet mode which prevents output being sent to the display console. The use of this option is highly recommended for all systems under load as the impact of writing lots of messages to the screen can seriously impact system throughput.

-r

An option for use when rolling log files are enabled to cause a new file to be started the first time a new event is logged each day. This functionality is enabled by adding the –r option (in conjunction with the –n option). This behavior applies to both text and PCAP format log files.

-x

This option is used to modify the filename format for rotating log files. By default the sequence number is appended at the end of the filename (eg. maint.log.2) but if the –x option is used the sequence number is placed before the file extension (eg. maint.2.log).

Examples

For example, the command line to run s7_log as module ID 0xef with rolling logs enabled would be:

```
s7_log -q -td -n20 -s1000 -m0xef -o0xff87 -fmaint.log
```

Typical output from s7_log is shown below:

```
S7L:2012-04-13 14:34:28.105 I0000 M t06a0 i0000 f20 def s60 p00100000
S7L:2012-04-13 14:34:28.187 I0000 LIU Status : id=0 SYNC LOSS
S7L:2012-04-13 14:34:28.187 I0000 LIU Status : id=0 AIS
S7L:2012-04-13 14:34:28.187 I0000 LIU Status : id=0 PCM LOSS
S7L:2012-04-13 14:34:28.187 I0000 LIU Status : id=0 REMOTE ALARM
S7L:2012-04-13 14:34:28.313 I0000 LIU Status : id=1 SYNC LOSS
S7L:2012-04-13 14:34:28.313 I0000 LIU Status : id=1 AIS
S7L:2012-04-13 14:34:28.313 I0000 LIU Status : id=1 PCM LOSS
S7L:2012-04-13 14:34:28.313 I0000 LIU Status : id=1 REMOTE ALARM
S7L:2012-04-13 14:34:28.516 I0000 Level 2 State : id=0 OUT OF SERVICE
S7L:2012-04-13 14:34:28.527 I0000 Level 2 State : id=1 OUT OF SERVICE
S7L:2012-04-13 14:34:28.805 I0000 LIU Status : id=0 AIS CLEARED
S7L:2012-04-13 14:34:29.004 I0000 LIU Status : id=1 AIS CLEARED
S7L:2012-04-13 14:34:29.504 I0000 Level 2 State : id=0 INITIAL ALIGNMENT
S7L:2012-04-13 14:34:29.505 I0000 Level 2 State : id=1 INITIAL ALIGNMENT
S7L:2012-04-13 14:34:30.006 I0000 Level 2 State : id=1 IN SERVICE
S7L:2012-04-13 14:34:30.006 I0000 Level 2 State : id=0 IN SERVICE
S7L:2012-04-13 14:34:30.008 I0000 MTP Event : linkset_id/link_ref=0100
Changeback
S7L:2012-04-13 14:34:30.008 I0000 MTP Event : linkset_id=01 Link set
recovered
S7L:2012-04-13 14:34:30.008 I0000 MTP Event : linkset_id=01 Adjacent SP
accessible
S7L:2012-04-13 14:34:30.008 I0000 MTP Event : point code=00000002
 Destination available
```

Each line of text corresponds to a received message. The parameter prefixed I is the instance recovered from the message. In an SIU host environment, the instance identifies the SIU (by the siu_id value) that originated the message. Instance 0 refers to SIUA and instance 1 refers to SIUB.

Messages that are not interpreted as text are displayed in hexadecimal format as follows:

```
M t<type> i<id> f<src> d<dst> s<status> e<err_info> p<param>
```

Each field contains the value of the corresponding message field in hexadecimal format.

7.3 s7_play

Description

s7_play is a utility, primarily intended for diagnostic purposes, which takes text based representation of messages and sends them to the DSI software environment. It can optionally wait for a response to a message, insert a delay between messages or pause until a specific message type is received.

Typically s7_play is used to prototype configuration sequences, or to generate status requests or statistics gathering messages from a live system.

Syntax

```
s7_play –v –m<module_id> -f<filename>
```

Parameters

-v

Show version information.

-m<module_id>

Set the module_id that s7_play will use. By default this is 0x5d but may need to be changed depending on the manner in which s7_play is being used. If s7_play is simply generating messages then it can run with the default module_id. If it is also receiving responses then it is important that there is a corresponding LOCAL entry in the system.txt file and that module_id is not in use by other processes. Also it is important that the correct module_id is entered in the src field of messages in the command file so that the responses come back to the correct message queue.

-f<filename>

The filename of the text file containing the commands to be executed by s7_play. Optionally a space may be inserted between –f and the file name. By convention the filename suffix .ms7 is used.

Example

For example, to run s7_play as module ID 0x2d and take commands from a file cmd.ms7.

```
s7_play –m0x2d –f cmd.ms7
```

7.3.1 s7_play Command File Format

The s7_play utility takes commands from a user-supplied text file and generates messages into the DSI software environment. This section details the format of the file and the syntax of all commands used within the file.

The s7_play command file is a text file with each line in the file representing a single command. The first character on the line determines the command type. Inserting a * or # character as the first character of a line causes the remainder of the line to be ignored by s7_play.

The following commands are supported:

Command	Function
M	Send message
W	Send message and wait for response
P	Pause and wait for a specified message type to be received
D	Delay
R	Change the receive message queue which s7_play uses when waiting for responses

The command file can be made self executing (within a Linux or Solaris environment) by using a feature of the Unix environment and including the following text (or similar) in the first line of the file and changing the file permissions to be executable. (Note however that this technique does not allow the module_id to be changed):

```
#!/opt/DSI/s7_play -f
```

M Command – Send Message

The send message command causes s7_play to allocate a message and populate it in accordance with the values contained within the file. S7_play then calls the GCT_send() function to send the message into the DSI software environment. The command format allows all fields of the message header, the parameter area and the message instance to be populated. Any fields not specified in the command are set by default to zero. As soon as the message has been sent, s7_play continues with the next command.

```
*
* The format for individual parameters is as follows:
*
*  -I0000 specifies the instance value for the message
*  -t0000 specifies the hdr->type value for the message
*  -i0000 specifies the hdr->id value for the message
*  -f00 specifies the hdr->src value for the message
*  -d00 specifies the hdr->dst value for the message
*  -r0000 specifies the hdr->rsp_req value for the message
*  -e00000000 specifies the hdr->err_info value for the message
*  -s00 specifies the hdr->status value for the message
*
* The param field is variable length up to 320 octets
*  -p0000..0000 specifies the param value for the message
*
* The following command sends a GEN_MSG_MOD_IDENT message to board_id=1
* NOTE: the message is a single line which wraps to fit the document!
*
M-I0001-t6111-i0000-fef-d8e-r8000-p00000000000000000000000000000000
00000000000000000000000
*
```

W Command – Send Message and Wait for Response

The wait command causes s7_play to allocate and send a message (as for the send message command). s7_play then reads messages from its own input message queue until it receives a response to the message it has just sent before continuing with the next command. When using this mode it is important that the src module_id is set (using the –f parameter) to the module_id of s7_play and that the appropriate bit in the rsp_req field is set so that a response is received. In addition it is important that the module_id in use by s7_play is not in use by another module.

```
*
* The following sends message type-0x1234 and waits for a response
W-t1234-i0000-sef-d8e-r8000-p000000
*
```

P Command – Pause until specific message received

The pause command causes s7_play to wait until a specific message type (as specified in the command) is received before continuing to the next command. The message type is a two byte hexadecimal value using the –t parameter.

```
*
* The following line pauses until message type 0x1234 is received
P-t1234
*
```

D Command – Delay for fixed interval

The delay command causes s7_play to delay for a (nominal) fixed interval before continuing to the next command. It takes two forms, one allowing the delay to be specified in seconds (using the –s parameter) and the other allowing the delay to be specified in milliseconds (using the –m parameter). The value of the delay is coded as a two byte hexadecimal value.

```
*
* The following line implements a 3 second delay
D-s0003
*
* The following line implements a 20ms delay
D-m0014
*
```

R Command – Change the Receive Message Queue

The "R" command causes s7_play to change the module_id that it uses for subsequent reads of its own message queue. The command takes a single 16 bit parameter designated –m as follows:

```
*
* The following line changes the s7_play module_id to 0x2d
R-m002d
*
```

7.4 tick and tim

Description

The tick and tim utilities are essential to the correct operation of any DSI deployment and both should always be started using the FORK_PROCESS command in the system.txt file.

The tick utility generates a periodic timing reference and sends it to the tim utility. The tim utility handles the timer mechanism for all other processes in the system, in most cases issuing a periodic timer tick message to the module every 100ms.

Syntax

```
tick [-v]
tim [-v]
```

Parameters

-v

Show version information.

Example

The following example shows the typical use of the tick and tim utilities as commands within the system.txt file:

```
FORK_PROCESS tim
FORK_PROCESS tick
```

7.5 s7_mgt

Description

The s7_mgt utility is the primary tool for configuring a DSI software stack. It is a single-shot configuration utility that takes configuration commands from a text file (config.txt by default).

The full set of configuration commands are detailed in section 8.16 of this manual.

As an alternative to using s7_mgt, experienced users can build their own configuration utilities using messaged-based configuration. In this case users should refer to the definitions of individual messages in the appropriate Programmer's Manuals.

Syntax

```
s7_mgt [-v -k<config_file> -m<module_id> -i<notify_id> -d -f<filename>]
```

Parameters

-v

Show version information.

-k<config file>

Specifies the filename of the user generated text file that contains all the protocol configuration commands. The default is config.txt.

-m<module id>

Run using an alternative specified module_id to the default. By default s7_mgt uses module_id=0xcf and typically this does not need to be changed.

-i<notify module id>

On completion of the single-shot configuration sequence, s7_mgt is able to generate a API_MSG_CNF_IND message to a user application indicating the completion status. The user application may use this indication to start up its own operation. This option is used to specify the module_id to which the notification message is sent. By default no notification is issued.

-d

Enables additional diagnostic output to assist diagnosis of configuration problems.

-f<filename>

Optionally specifies a text file to which the output from s7_mgt will be written. s7_mgt will overwrite existing log files.

Example

The following example uses the configuration file "my_config.txt" and on completion will issue notification to module_id=0xef.

```
s7_mgt -kmy_config.txt -i0xef
```

7.6 ssd

ssd is the generic name for the process that interfaces with the per-board device driver for passing messages to and from the board. It also controls resetting and downloading software onto the board.

ssd also provides the ability to configure the addressing mode for the board, this is particularly important where multiple boards of the same type are in use in a single server to ensure that the board_id always refers to the same board.

Each board type has its own version of ssd as follows:

ssds for SPCI4/SPCI2S boards
ssdl for SS7LD boards
ssdh for SS7HD boards
ssdm for SS7MD boards

7.6.1 ssds (for SPCI4/SPCI2S boards)

Description

The ssds utility interfaces with the device driver for passing messages to and from the SPCI4 and SPCI2S boards and controls the downloading software to the board. ssds also controls geographic addressing for all boards in a system which can be based on either the PCI bus enumeration or the ADDR switch setting on the board.

Syntax

```
ssds [-v -o -a –d -m]
```

Parameters

-v

Show version information.

-o<addressing mode>

This parameter specifies the Geographic Addressing mode of operation. Geographic Addressing allows the logical position of a board (or board_id) in a system to remain the same irrespective of the addition or removal of other boards on the PCI bus. It can take the following values:

-o1 - PCI address mode where address is determined by enumerating boards on the PCI bus at boot time (i.e., the default order found by the operating system).

-o3 - Switch address mode where address is determined by a 16 position ADDR switch on the board. The switch must be set to a different value for each board.

If the parameter is omitted then operation defaults to PCI address mode.

-a<address>

For Switch address mode it is necessary to specify a second command line parameter s containing a list of the addresses for each logical board position (or board_id) derived from the ADDR switch setting. Each entry in the list (up to a maximum of 16) is separated by a comma as follows:

```
-a6,4,2,3,12,14
```

If using Switch address mode , board_id=0 would be the board with ADDR switch set to 6, board_id=1 would be the board with ADDR switch set to 4 and so on. It is not necessary for all boards listed in this option to physically exist in a system.

-d

Enables additional diagnostic output to provide feedback on progress of code file download and initialization to help resolve configuration issues.

-m<module id>

Run using an alternative specified module_id to the default. By default ssds uses module_id=0x20.

7.6.2 ssdl (for SS7LD boards)

Description

The ssdl utility interfaces with the device driver for passing messages to and from the SS7LD board and controls the downloading software to the board. ssdl can be configured to handle different modes of addressing for each board within a system. This can be based on either the PCI bus enumeration or board serial number.

Syntax

```
ssdl [-v –o -a -d –m –Lp –Lt -t]
```

Parameters

-v

Show version information.

-o<addressing mode>

This parameter specifies the Geographic Addressing mode of operation. Geographic addressing allows a board's logical position in a system to remain the same irrespective of the addition or removal of other boards on the PCI bus. Two different schemes of addressing DSI SS7LDH4Q boards are supported:

-o1 - PCI address mode, as supplied by enumerating boards on the PCI bus at boot time

-o3 - Switch address mode based addressing, determined by a 16-position rotary switch (SW1) on the board. Note that any changes to the ADDR switch setting will not be recognized by the system until the system is power cycled.

If the parameter is omitted, then operation defaults to PCI address mode.

-a<address>

For Switch address mode, it is necessary to specify a second command line parameter containing a list of the switch settings for each logical board position (or board_id). Each entry in the list (up to a maximum of 16) is separated by a comma as follows:

```
-a6,4,2,3
```

If using Switch address mode, board_id=0 would be the board with ADDR switch set to 6, board_id=1 would be the board with ADDR switch set to 4, and so on. It is not necessary for all boards listed in this parameter to actually exist in a system. A board that is listed, but missing, would result in a gap in the logical board_id sequence.

-d

Enables additional diagnostic output to provide feedback on progress of code file download and initialization to help resolve configuration issues.

-m<module id>

Run using an alternative specified module_id to the default. By default ssdl uses module_id=0x20.

-Lp<license path>

Specifies the path to the license file.

-Lt

License test mode option used to check that the specified license is valid. The result is displayed to the console.

-t

Permits the module to run without a license in 'trial mode' for one hour after which the binary will terminate.

7.6.3 ssdh (for SS7HD boards)

Description

The ssdh utility interfaces with the device driver for passing messages to and from the SS7HD board and controls the downloading software to the board. ssdh handles different modes of addressing for boards within a system. This can be based on either PCI bus enumeration or the ADDR switch setting on the board.

Syntax

```
ssdh [-v -o -a -d –m]
```

Parameters

-v

Show version information.

-o<addressing mode>

Select geographic address mode. Geographic addressing allows a board's logical position in a system to remain the same irrespective of the addition or removal of other boards on the PCI bus. The following addressing schemes are supported:

-o1 – PCI address mode as supplied by enumerating boards on the PCI bus at boot time

-o3 - ADDR switch-based addressing, determined by a 16-position rotary switch on the board

If the parameter is omitted then operation defaults to PCI address mode.

-a<address>

For switch based addressing, it is necessary to specify a second option that provides a list of the switch settings to be used for each logical board position (or board_id).

```
-a6,4,2,3,12,14
```

Up to a maximum of 16 addresses can be specified in this list. In the example above, board_id = 0 would be the board with the ADDR rotary switch set to position 6, board_id = 1 would be the board with the rotary switch set to position 4 and so on.

It is not necessary for all boards listed in this option to physically exist in a system.

-d

Enables additional diagnostic output to provide feedback on progress of code file download and initialization to help resolve configuration issues.

-m<module id>

Run using an alternative specified module_id to the default. By default ssdh uses module_id=0x20.

Example

The following example shows the use of a three-board system using the hardware switch mode where the switches would be set to 1 for board_id=0, 2 for board_id=1 and 5 for board_id=2.

```
ssdh -o3 -a1,2,5
```

7.6.4 ssdm (for SS7MD boards)

Description

The ssdm utility interfaces with the device driver for passing messages to and from the SS7MD board and controls the downloading software to the board. ssdm can be configured to handle different modes of addressing for each board within a system. This can be based on either the PCI bus enumeration or board serial number.

Syntax

```
ssdm [-v -o -a -d –m –Lp –Lt –t]
```

Parameters

-v

Show version information.

-o<addressing mode>

Select geographic address mode. Geographic addressing allows a board's logical position in a system to remain the same irrespective of the addition or removal of other boards on the PCI bus. Two different addressing schemes are supported:

-o1 – PCI address mode, as supplied by enumerating boards on the PCI bus at boot time

-o2 - Board serial number, determined by the board unique serial number

If the parameter is omitted then operation defaults to PCI address mode.

-a<address>

For serial number based addressing, it is necessary to specify a second option that provides a list of the serial numbers of the board to reside at each logical board location. Up to a maximum of eight addresses can be specified in the following format:

```
-aPX800020,PX800015,PX800015,PX801000
```

It is not necessary for all boards listed in this option to physically exist in a system. In board serial number address mode, if a board does not have a valid entry in the address list, that board will be inaccessible to the system.

To leave a logical board id unused then a dummy entry (e.g. PX800000) should be included in that position in the address list.

Under certain circumstances (for example to determine the serial number of a new board added to the system which, as yet, does not have a valid mapping in the system.txt file), the user may require access to all the boards in a system irrespective of the address mode or any address list specified in the system.txt file.

To retrieve a board's serial number under these conditions, the SSD_MSG_BOARD_INFO message allows each board to be addressed either via its logical address (as determined by the address list mapping) or via its physical address (as determined via its discovery order in the platforms PCI bus enumeration). To access the board under its physical address, the most significant bit of the SSD_MSG_BOARD_INFO ID field is set.

-d

Enables additional diagnostic output to provide feedback on progress of code file download and initialization to help resolve configuration issues.

-m<module id>

Run using an alternative specified module_id to the default. By default ssdm uses module_id=0x20.

-Lp<license path>

Specifies the path to the license file.

-Lt

License test mode option used to check that the specified license is valid. The result is displayed to the console.

-t

Permits the module to run without a license in 'trial mode' for one hour after which the binary will terminate.

Example

The following example is for a two-board system using the board serial number address mode where serial numbers PX800007 and PX800046, map to board identifiers 0, and 1, respectively:

```
ssdm -o2 -aPX800007,PX800046 -Lpc:/opt/LIC
```

7.7 rsi

Description

The RSI utility allows two DSI environments operating on separate platforms to extend the message passing mechanism to work between the two platforms over TCP/IP. The RSI utility includes mechanisms to detect link failure and manage link restoration. The RSI utility creates one instance of the rsi_lnk process for each RSI link that is created up to a maximum of 32 links.

RSI is the primary means by which user applications interface with the Dialogic® DSI Signaling Interface Unit, in this case the SIU is the server end of the RSI link. RSI can also be used for generic communication between two host based DSI environments.

Syntax

```
rsi –v
rsi –m -p<pipe> -l<link_selection>-r<link_process> -nl
```

Parameters

-v

Show version information.

-m<module id>

Run using an alternative specified module_id to the default. By default rsi uses module_id=0xb0.

-p<pipe>

Specifies the pipe used for communication between rsi and rsi_lnk. If not specified, rsi attempts to use `/tmp/pipe`.

This parameter is not used when operating under Windows.

-l<link_selection>

Specifies the algorithm to be used by RSI to select which RSI link to use for sending a message. Messages are routed according to their Instance value (which is set by the sending module using the GCT_set_instance() function) and the link selection algorithm. The following algorithms are supported:

Value	RSI Selection Algorithm
1	Messages are routed by Instance value contained within the message. This allows the sending application to directly select which link will be used to send a message. It is the default and most widely used algorithm.
2	All messages are routed to rsi_link_id=0.
3	The message is sent on the lowest available (and in service) rsi_link_id.

-r<link_process>

Specifies the location of the rsi_lnk binary. If not specified, rsi assumes that the rsi_lnk binary is located in the current directory.

-nl

Enables transmission of long messages.

Example

Example rsi entry in the system.txt file:

```
FORK_PROCESS ../BIN/rsi -r../BIN/rsi_lnk -l1
```

7.8 rsicmd

Description

The rsicmd utility is a command line utility to configure an individual RSI link.

Syntax

```
rsicmd <link_id> <conc_id> <link_type> <IP_addr> < port_number>
```

Parameters

<link_id>

The local logical RSI link identifier in the range 0 to one less than the number of links supported.

When connecting from a host to a pair of SIUs in a dual redundant configuration, rsi_link_id=0 should be used to communicate with SIUA and rsi_link_id=1 should be used to communicate with SIUB.

<conc_id>

The concerned module_id to which RSI link status indications will be sent.

<link_type>

Client / Server setting. This bit should be set to 0 for the Client end of the link and set to 1 for the Server end of the link. All SIU Host links to an SIU must be created as Client link types.

<IP_addr>

The IP address (IPv4 and IPv6 address formats are supported). For the Client end of the link this should be set to the IP address of the remote machine. For the Server end of the link this should be set to the machine's local IP address.

<port number>

Specifies the TCP/IP port number used for the RSI link. For SIU Hosts the first host (host_id=0) should connect to port number 9000. Subsequent hosts connect to ports 9001, 9002 etc.

Examples

For example, to start a link to SIUA with an IPv4 address 123.124.125.126 as host 0, nominating a module whose ID is 0xef to receive RSI status information, the command line is:

```
rsicmd 0 0xef 0 123.124.125.126 9000
```

The following IPv6 address formats, as specified by RFC4291 and RFC4007, are supported:

```
* IPv6 link local address via eth0
rsicmd 1 0xef 1 fe80::20d:60ff:feb7:d751%eth0 9000

* IPv6 global address
rsicmd 1 0xef 1 fd77:19a9:8cf0:0:20d:60ff:feb7:d751 9000

* IPv4 or IPv6 via DNS lookup
rsicmd 1 0xef 1 dpkbuild.lab.companyname.com 9000
```

rsicmd may be run from system.txt by adding the appropriate FORK_PROCESS commands, hence to connect to both SIUA and SIUB as host ID 3, the following commands would be entered in the system.txt file on the host:

```
FORK_PROCESS ..\RUN\rsicmd 0 0xef 0 123.234.345.456 9003
FORK_PROCESS ..\RUN\rsicmd 1 0xef 0 123.234.345.457 9003
```

7.9 tempmon

Description

The tempmon utility periodically monitors the operating temperature of SS7MD and SS7LD boards to support evaluation of a suitable host chassis for deployment. The utility runs directly above the board driver and does not require or make use of the GCT environment.

The tempmon output which can optionally be sent to file includes date and time of all readings and the serial number of all boards detected.

The tempmon utility can be shut down by pressing <CTRL>C. The application will then close any log file and exit.

Syntax

```
tempmon [-v [-f<filename>] [-t<sample period>] [-b<board mask]
```

Parameters

-v

Show version information.

-f <filename>

Optionally specifies a file to which all output is written.

-t <sample period>

Period, in seconds, between successive temperature readings. Defaults to 1 second.

-b <board mask>

A 16 bit bitmap of boards to include (each bit set will display that board). The least significant bit corresponds to board_id=0. If the parameter is omitted a default value of 0x000f is used which will display the temperature for the first 4 boards.

Example

```
tempmon -ftemplog.txt -t5
```

Sample Output

```
tempmon: Temperature monitor (C) 2009 Dialogic Corporation
============================================================
2009-06-02 10:36:00, PX800007, PX800046, PX800057, PX800023
2009-06-02 10:36:00,       35,       36,       34,       35
2009-06-02 10:36:05,       35,       36,       34,       35
2009-06-02 10:36:10,       35,       36,       35,       36
2009-06-02 10:36:15,       35,       37,       35,       36
2009-06-02 10:36:20,       35,       37,       35,       37
2009-06-02 10:36:25,       35,       37,       35,       37
2009-06-02 10:36:30,       35,       38,       35,       37
2009-06-02 10:36:35,       35,       38,       35,       37
```

7.10 dsictrl

Description

The dsictrl utility is a command line utility that allows the user to perform interactive control to the various elements within a DSI deployment.

dsictrl supports control of the following entities: E1/T1 LIUs, MTP links, SIGTRAN links, Call control circuit groups and RSI links.

MTP links can be Activated, Deactivated, Inhibited & Uninhibited.

Circuit groups can be Reset, Maintenance or Hardware Blocked and Unblocked.

LIUs can be forced to generate alarm conditions (eg RAI, AIS) and put in various loopback modes.

For a full syntax listing run the tool with the –h option.

Each invocation of dsictrl performs an action on a single element. In order to perform operations on multiple elements users should create a script file containing a separate invocation of dsictrl on each line of the file.

Syntax

```
dsictrl <type> <action> <id> [-m -dm -di]
dsictrl -h
```

Parameters

<type>

A token indicating the type of entity being acted upon as detailed in the following table:

<type>	Description	LIU	MTPL	M3UAL	CGRP	RSIL
LIU	Line Interface Unit.	•				
MTPL	MTP Signaling Link		•			
M3UAL	M3UA SIGTRAN Link			•		
CGRP	Call Control Circuit Group				•	
RSIL	RSI Link					•

<action>

A token indicating the action to be taken as detailed in the following table:

<action>	Description	LIU	MTPL	M3UAL	CGRP	RSIL
ACT	Activate		●			●
DEACT	Deactivate		●			●
INH	Maintenance Inhibit		●			
UNINH	Maintenance Uninhibit		●			
GRS	Reset Circuit Group				●	
MCGB	Maintenance Block Circuit group				●	
MCGU	Maintenance Unblock Circuit group				●	
HCGB	Hardware Block Circuit Group				●	
HCGU	Hardware Unblock Circuit Group				●	
AIS	Force generate of AIS (Blue Alarm)					●
NOAIS	Cancel forced generation of AIS (Blue Alarm)					●
RAI	Force generation of RAI (Yellow Alarm)					●
NORAI	Cancel generation of RAI (Yellow Alarm)					●
AUTORAI	Set RAI (Yellow Alarm) generation to automatic					●
RLOOP	Activate remote loopback					●
LLOOP	Activate local loopback					●
NOLOOP	Cancel all loopbacks					●

<id>

The identifier indicating the instance of the entity to be controlled. The id is formatted according to the following table:

Format of <id>	Description	LIU	MTPL	M3UAL	CGRP	RSIL
x	The liu_id within a board in the range from 0 to one less than the number of LIUs on the board. (Note: When using multiple boards, the board_id must be specified using the –di parameter).	●				
x-y	The MTP Signaling Link id where x is the linkset_id and y is the link_ref.		●			
x	The SIGTRAN link_id.			●		
x	The Circuit Group Identifier (gid) in the range 0 to one less than the number of groups supported.				●	
x	The RSI link_id.					●

-m<module id>

Run using an alternative specified module_id to the default. By default dsictrl uses module_id=0x3d.

-dm<dest_module_id>

The optional destination module id. Default destination module_ids for each entity can be listed using the –h option.

-di<dest_module_instance>

The optional destination module instance, used for example when communicating with multiple boards to specify the board_id. If not specified, –di defaults to 0.

Examples

```
dsictrl MTPL ACT 0-1
dsictrl MTPL ACT 1-1 -dm0x82
```

7.11 dsistat

Description

The disstat utility is a command line utility that allows the user to request and display status and measurements from the various elements within a DSI deployment.

For a full syntax listing run the tool with the –h option.

Syntax

```
dsistat <type> <action> <id> [-r -m -dm -di -sh -sr]
dsistat -h
```

Parameters

<type>
A token indicating the type of element for which the status or measurements are to be read. For example: MTPL, MTPR, CGRP, CCTS, LIU, SCTPP, SCTPA, M3UAP, M3UAS, M3UAS, M3UAR, LSS, RSS, RSP, RSIL. For full details run the dsistat with the –h option.

<action>
A token which should be set to STATUS to read back status or STATS to read back measurements.

<id>
The identifier of the element. For full details of the available options and the format of the parameter run dsistat using the –h option.

-r
An optional parameter to cause the measurements to be reset.

-m<module id>
Run using an alternative specified module_id to the default. By default dsistat uses module_id=0x3d.

-dm<dest_module_id>
The optional destination module id. Default destination module_ids for each entity can be listed using the –h option.

-di<dest_module_instance>
The optional destination module instance, used for example when communicating with multiple boards to specify the board_id. If not specified, –di defaults to 0.

-sh
Optional parameter causes the short format of the output to be displayed omitting the header. This is useful when creating a script to read status or measurements from several elements. The header is only needed for the first line and subsequent invocations of dsistat can use the –sh option.

-sr

Optional parameter causes the short format of the output to be displayed omitting the status footer. This is useful when creating a script to read status or measurements from several elements. The footer may not be required.

Example

The following are examples of individual commands:

```
dsistat MTPL STATUS 1-0
dsistat MTPR STATS 1232 -r -dm0x82
dsistat RSIL STATS 0
```

An example of a script file which displays a header for the first row and lists status only in subsequent rows is shown below:

```
dsistat rsp status 639 -sr
dsistat rsp status 756 -sh -sr
dsistat rsp status 9064 -sh
```

Example output from the above script is shown below:

```
SPC         STATE
639         ALLOWED
756         PROHIBITED
9064        ALLOWED
Executed
```

7.12 dsitrace

Description

The distrace utility is a command line utility that allows the user to conveniently set the trace masks for individual protocols from the command line.

For a full syntax listing run the tool with the –h option.

Syntax

```
dsitrace <type> <action> [-ti -to -tm -m -dm -di]
dsitrace -h
```

Parameters

<type>
A token indicating the protocol module (eg MTP3, ISUP, SCCP, TCAP, MAP, INAP, IS41, M3UA, SUA, SCTP, AAL5).

<action>
A token which should be set to TRACE to activate tracing or NOTRACE to deactivate tracing.

-m<module id>
Run using an alternative specified module_id to the default. By default dsitrace uses module_id=0x3d.

-dm<dest_module_id>
The optional destination module id. Default destination module_ids for each entity can be listed using the –h option.

-di<dest_instance>
The optional destination module instance, used for example when communicating with multiple boards to specify the board_id. If not specified, –di defaults to 0.

-ti<input_event_mask>
The value to use for the input event mask. This parameter is optional and when not specified dsitrace will select a per-module default value. The default value can be listed by running dsitrace with the –h option.

-to<output_event_mask>
The value to use for the output event mask. This parameter is optional and when not specified dsitrace will select a per-module default value. The default value can be listed by running dsitrace with the –h option.

-tm<mgmt_event_mask>
The value to use for the management event mask. This parameter is optional and when not specified dsitrace will select a per-module default value. The default value can be listed by running dsitrace with the –h option.

Examples

```
dsitrace MTP3 TRACE
dsitrace MTP3 TRACE -ti0x00000003 -dm0x82
```

8 Configuration Command Reference

This section describes the commands and parameters used in the protocol configuration file config.txt. These are used by the s7_mgt utility to perform single-shot configuration of the protocol stack at startup.

8.1 Physical Interface Configuration Commands

The physical interface configuration commands are:

- SS7_BOARD Command
- LIU_CONFIG Command
- LIU_SC_DRIVE Command
- SCBUS_LISTEN Command
- STREAM_XCON Command (Cross Connect Configuration)

8.1.1 SS7_BOARD Command

Synopsis

Command to configure an SS7 board in the system

Syntax

SS7_BOARD <board_id> <board_type> <flags> <code_file> <run_mode>

Example

SS7_BOARD 0 SPCI4 0x0043 ./DC/ss7.dc3 ISUP-L

Parameters

<board_id>

The logical identity of the board in the range from 0 to one less than the number of boards supported.

<board_type>

The board type within the system. Possible values are:

SPCI2S, SPCI4,
SS7HDP, SS7HDE,
SS7MD,
SS7LD,
DNI2410, DNI1210, DNI610 and DNI310

<flags>

A 32 bit value that provides additional level 1 configuration for the board. The meaning of each bit may vary with different board types. The bits in the flags field are used as follows (when using a DNIxxxx board the flags should be set to zero):

Flag Bit	Support			
	SPCI	SS7HD	SS7MD	SS7LD
0	•	•	•	•
1	•	•	•	•
3		•		
6	•	•		•
7	•	•		•
9	•	•	•	•
13	•	•		•

Bit 0 controls the reference source used for on-board clocks when acting as CT bus Primary Master. If set to 1 then the clock is recovered from one of the line interfaces. If set to zero then the on-board clock oscillator is used.

Bit 1 is reserved for future use and must always be set to 1.

Bit 3 is applicable for the SS7HDP and SS7HDE boards only. It should be set to 1 to enable H.100 bus termination or set to 0 to disable H.100 bus termination. Setting bus termination prevents the bus clock signal from being reflected and must be set for any board at either end of the H.100 bus. For all other board types this bit should be set to 0.

Bit 6 and **7** together select the initial clocking mode for the CT bus as shown in the following table. The clocking mode can subsequently modified dynamically using the MVD_MSG_CNFCLOCK message:

Bit 7	Bit 6	CT Bus Clocking Mode
0	0	**The CT bus interface is disabled -** In this mode, the board is electrically isolated from the other boards using the CT bus. The CT bus connection commands may still be used, but the connections made are only visible to this board. When using this mode, the on-board clocks are synchronized to the source selected by bit 0 of this flags parameter.
0	1	**Primary Master, Clock set A** - The board drives CT bus clock set A using the clock source selected by bit 0 of this flags parameter.
1	0	**Secondary Master, Clock set B** - The board is configured to drive clock set B in Secondary Master mode. It automatically switches to become Primary Master if the board driving clock set A fails. While acting as Secondary Master the on-board clocks are synchronized to the CT bus clock set A.
1	1	**Slave, initially using Clock set A** – The board uses the CT bus clocks, which must be generated by another board on the CT bus. Initially the board recovers from clock set A, though will switch over automatically to recover from clock set B if set A fails.

Bit 9 – Typically this bit is not used and should be set to 0. In dual board fault tolerant configurations where the MTP3 layer is running on the board, Board A must set bit 9 to 0 while Board B must set bit 9 to 1.

Bit 13 causes the board to drive the CT_NETREF1 clocks on the CT bus when set to 1. The highest priority in-sync line interface is used as a clock source. If this bit is set to zero then the CT_NETREF1 clock is not driven. By default, liu_id=0 is the highest priority and liu_id=7 is the lowest. The priority may however be modified using the MVD_MSG_CLOCK_PRI message.

Bit 16 is set to 1 to enable SNMP on a per-board basis. Information provided through this configuration includes board specific data, and all Line Interface Units subsequently configured. For additional information on SNMP support refer to the *Dialogic® DSI Protocol Stacks SNMP User Manual*.

All other bits are reserved and must be set to zero.

<code file>

The filename of the Code File which gets downloaded to the board when it is reset. To support code file paths the code file name may be up to 49 characters long. Each board family uses a different file suffix as follows:

Board Family	Code File Suffix
SPCI	.dc3
SS7HD	.dc4
SS7MD	.dc6
SS7LD	.dc7

All appropriate SS7 protocols for the board are included within the Code File. The selection of which protocols are run is made using the run_mode parameter below.

When using DNIxxxx boards <code_file> is not used and should be set to 'null'.

<run_mode>

The run_mode determines which protocols are invoked at run time. The different board families have separate run modes. For details on what is supported refer to the Programmer's Manual for your specific board.

When using SS7LD or DNIxxxx boards, <run_mode> should be set to one of the following values depending on which protocols are required to run as part of the ssdl module. If not running within ssdl then the user can run the protocol as a stand-alone host binary. All protocols that run embedded within ssdl use their own message queue so they require a LOCAL entry in the system.txt file.

Run Mode	Protocols running embedded within ssdl	Optional Host Protocols
MTP2	None	MTP3, ISUP, SCCP etc
MTP	MTP3	ISUP, SCCP etc
ISUP	MTP3 and ISUP	SCCP etc

8.1.2 LIU_CONFIG Command

Synopsis

This command configures the operating parameters for a T1/E1 Line Interface Unit (LIU).

Syntax

LIU_CONFIG <board_id> <liu_id> <liu_type> <line_code> <frame_format> <crc_mode> [<build_out> <options>]

Example

LIU_CONFIG 0 0 5 1 1 1 0x0000

Parameters

<board_id>

The logical identity of the board in the range from 0 to one less than the number of boards supported.

<liu_id>

The identifier of the T1/E1 Line Interface Unit in the range from 0 to one less than the number of LIUs supported (except for the SPCI2S board where the valid values are 2 and 3).

<liu_type>

The physical type of interface according to the following table:

liu_type	Description	Support			
		SPCI	SS7HD	SS7MD	SS7LD
1	Disabled (used to deactivate a LIU). In this mode the LIU does not produce an output signal.	•	•	•	•
3	E1 120ohm balanced interface.	•	•	•	•
4	T1	•	•	•	•
5	E1 Impedance according to hardware	•	•	•	•
6	E1 high-impedance (for monitoring applications)		•	•	•
7	T1 high-impedance (for monitoring applications)		•	•	•
8	E1 PMP mode (for monitoring with a Protected Monitoring Point)			•	•
9	T1 PMP mode (for monitoring with a Protected Monitoring Point)			•	•

Note: When using the SS7LD board all four ports must be configured as either T1 or E1.

<line_code>

The line coding technique taken from the following table:

line_code	Description	Support			
		SPCI	SS7HD	SS7MD	SS7LD
1	HDB3 (E1 only).	•	•	•	•
2	AMI with no Zero Code Suppression.	•	•	•	•
3	AMI with Zero Code Suppression (The appropriate bit in the clear_mask parameter may be set to disable Zero Code Suppression for individual timeslots if required) (T1 only).	•	•		•[3]
4	B8ZS (T1 only).	•	•	•	•

<frame_format>

The frame format taken from the following table:

frame_ format	Description	Support			
		SPCI	SS7HD	SS7MD	SS7LD
1	E1 double frame (E1 only)	•	•	•	•
2	E1 CRC4 multiframe (E1 only)	•	•	•	•
3	F4 4-frame multi-frame (T1 only)			•	
4	D3/D4 (Yellow alarm = bit 2 in each channel) (T1 only)	•	•	•	•
7	ESF (Yellow alarm in data link channel) (T1 only)	•	•	•	•
8	F72/SLC96 (72-frame multi-frame) (T1 only)			•	
9	J1 frame format. (<liu_type> must be 4; T1)		•	•	
10	Unstructured mode (HSL)		•		

<crc_mode>

The CRC mode. The following table shows the permitted values and their meanings.

[3] Note: The ability to disable Zero Code Suppression on a per timeslot basis is not supported by the SS7LD board.

crc_mode	Description	Support			
		SPCI	SS7HD	SS7MD	SS7LD
1	CRC generation disabled	•	•	•	•
2	CRC4 enabled (frame_format must be set to 2)	•	•	•	•
3	CRC4 compatibility mode (frame_format must be set to 2)	•	•		
4	CRC6 enabled (frame_format must be set to 7)	•	•	•	•
5	CRC4 G.706 mode (frame_format must be set to 2) Note: Out of CRC4-multiframe E-Bits are transmitted as zeroes.		•		

<build_out>

The build out type. The following table shows the permitted values and their meanings.

Value	Description	Valid For	Support			
			SPCI	SS7HD	SS7MD	SS7LD
0	E1 setting (default)	liu_type = 3 or 5	•	•	•	•
1	T1 short haul, 0 to 110 ft. (default)	liu_type = 4		•	•	•
2	T1 short haul, 0 to 110 ft. (same as value=1)			•	•	•
3	T1 short haul, 110 to 220 ft.			•		•
4	T1 short haul, 220 to 330 ft.			•		•
5	T1 short haul, 330 to 440 ft.			•		•
6	T1 short haul, 440 to 550 ft.			•		•
7	T1 short haul, 550 to 600 ft.			•		•
8	T1 long haul LB0 (-0db)			•	•	•
9	T1 long haul LB0 (-7.5db)			•	•	
10	T1 long haul LB0 (-15db)			•	•	
11	T1 long haul LB0 (0db, TR62411)			•	•	

<options>

A 16 bit value that provides additional per-LIU run-time configuration options. The bits in the **<options>** field are used as follows:

Bit 0 set to 1 to prevent the LIU being used as a source of clock recovery. This option is applicable only for the SS7MD board. For SPCI and SS7HD boards this functionality can be achieved by using the LIU priority message.

All other bits are reserved and must be set to zero.

8.1.3 LIU_SC_DRIVE Command

Synopsis

This command is used during initialization to set up a static switch path through the board between the Line Interface Unit (LIU) and the CT bus. It connects selected incoming voice timeslots from an T1/E1 LIU to a sequential block of channels on the CT bus and prepares the outgoing timeslots for subsequent use by the MVD_MSG_SC_LISTEN message.

Note: This command is only supported for SPCI and SS7HD product families.

Syntax

LIU_SC_DRIVE <board_id> <liu_id> <sc_channel> <ts_mask> {<mode>}

Example

LIU_SC_DRIVE 0 0 512 0xfffefffe

Parameters

<board_id>

The logical identity of the board in the range from 0 to one less than the number of boards supported.

<liu_id>

The identifier of the T1/E1 Line Interface Unit in the range 0 to one less than the number of LIUs supported (except for the SPCI2S board where the valid values are 2 and 3). This parameter can also be set to special values which are board specific.

For SPCI boards, value 0x83 selects the signaling processor instead of an LIU. In this case timeslots 0 ... 3 in the ts_mask correspond to signaling processor 0…3.

For SS7HD boards, this parameter can also be set to one of the special values 0x90, 0x91, 0x92, and 0x93, depending on the number of signaling processors. In these cases, the timeslots 0 to 31 in the <ts_mask> parameter correspond to the signaling processor's signaling links.

<sc_channel>

The channel number of the first channel to be used on the CT bus. This must be in the range from 0 up to one less than the total number of channels on the CT bus.

<ts_mask>

A 32 bit timeslot mask where each bit position is set to 1 if the corresponding timeslot on the T1/E1 interface is required to be connected to the CT bus. The least significant bit (bit 0) represents timeslot 0. Each timeslot for which the corresponding bit is set in ts_mask is connected up to the CT bus; other timeslots are not affected in any way.

Timeslots containing SS7 signaling that are processed by the signaling processor on the board should not be included in the timeslot mask. Usually the mask should be set to include all bearer (voice) timeslots but no signaling timeslots. Bit 0 (corresponding to timeslot 0 on the LIU) must not be set.

As an example, for an E1 interface with SS7 signaling on timeslot 16, and the remaining 30 timeslots used for voice circuits, ts_mask should be set to the value 0xfffefffe. For a T1 interface with signaling on timeslot 24, ts_mask should be set to the value 0x00fffffe.

<mode>

This parameter allows the user to select how the CT bus channels are allocated. Usually (mode=1) the first timeslot connected to the CT bus is connected to sc_channel and each subsequent timeslot that is selected is connected to the next CT bus channel. This allows maximum utilization of channels on the CT bus.

An alternative mode (mode=2) (only used if there is a specific requirement for it) associates (but does not necessarily connect) timeslot 0 on the LIU with sc_channel and subsequent timeslots on the LIU with subsequent CT bus channels. Connections are only made when the corresponding bit in the timeslot mask is set to 1. This mode of operation preserves the spacing between timeslots that was originally found on the T1/E1 interface but does result in a number of CT bus channels being not used.

The mode parameter is optional and may be omitted altogether. This has the same effect as setting it to 1.

8.1.4 SCBUS_LISTEN Command

Synopsis

This command establishes a connection from the CT bus to an outgoing timeslot on the Line Interface Unit (LIU).

Dynamic modification of voice paths can only be performed by issuing messages directly to the board. The MVD_MSG_SC_LISTEN message is recommended for this purpose.

Note: This command is only fully supported for SPCI and SS7HD product families. For the SS7MD board this command can be used to switch between timeslots between LIUs on the same board. Refer to SS7MD Programmer's Manual for full details.

Syntax

SCBUS_LISTEN <board_id> <liu_id> <timeslot> <sc_channel>

Example

SCBUS_LISTEN 0 0 31 23

Parameters

<board_id>

The logical identity of the board in the range from 0 to one less than the number of boards supported.

<liu_id>

The identifier of the T1/E1 Line Interface Unit in the range 0 to one less than the number of LIUs supported (except for the SPCI2S board where the valid values are 2 and 3). This parameter can also be set to special values which are board specific.

For SPCI boards, value 0x83 selects the signaling processor instead of an LIU. In this case timeslots 0 ... 3 in the ts_mask correspond to signaling processor 0…3.

For SS7HD boards, this parameter can also be set to one of the special values 0x90, 0x91, 0x92, and 0x93, depending on the number of signaling processors. In these cases, the timeslots 0 to 31 in the <ts_mask> parameter correspond to the signaling processor's signaling links.

<timeslot>

The timeslot number on the T1/E1 line interface unit on which the data from the CT bus is transmitted. The valid ranges for timeslot are 1 to 31 for an E1 interface, 1 to 24 for a T1 interface and 0 to 31 when referring to the signaling processor on the SS7HD board.

<sc_channel>

The channel number on the CT bus to which the LIU listens. This must be in the range from 0 up to one less than the total number of channels on the CT bus.

8.1.5 STREAM_XCON Command (Cross Connect Configuration)

Synopsis

The STREAM_XCON command controls the cross connect switch on the signaling boards, enabling the cross connection of timeslots between two Line Interface Units (LIUs) on each signaling board. The LIUs on a board are referenced by a fixed logical stream number.

This command is supported for SPCI, SS7HD and SS7LD boards.

Syntax

STREAM_XCON <bpos> <stream_a> <stream_b> <mode> <ts_mask> <pattern>

Example

STREAM_XCON 3 2 3 3 0xfffefffe 0

Parameters

<bpos>

The board position of the cross connect switch to be controlled. There must be a valid board at this position (previously defined by an SS7_BOARD command).

<stream_a>

Reference to the 2 Mb/s stream for the output of the connection. There must be a valid LIU at this position (previously defined by a LIU_CONFIG command). Valid values are:

Dialogic® DSI SS7 Board Type	Stream	T1/E1 Interface
SPCI4, SS7HD, SS7LD	0	L1
	1	L2
SPCI2S, SPCI4, SS7HD, SS7LD	2	L3
	3	L4

<stream_b>

A reference to the 2 Mb/s stream for the input of a simplex connection (mode 2) or one half of a duplex cross connection (mode 3). In other modes, this field should be set to 0. There must be a valid LIU at this position (previously defined by a LIU_CONFIG command). For valid values, see the table in the <stream_a> parameter description above.

<mode>

Indicates the requested cross connect switch function according to the following table.

Mode	Function
1	Not supported
2	Connect the input timeslot to the output timeslot
3	Duplex cross-connect the input and output timeslot

<ts_mask>

A 32-bit mask specifying the timeslots to which the cross connect is applied to. Each bit corresponds to a timeslot in the input/output stream. Bit 0 (the least significant bit) corresponds to timeslot number 0. To apply this command to a timeslot, the corresponding bit must be set to one.

— E1 interfaces have 32 timeslots numbered 0 to 31. Timeslot 0 is used for frame alignment and timeslot 16 is generally used for signaling or is empty. Hence the normal configuration is to cross connect timeslots 1 to 15 and 17 to 31 between the two ports on each signaling board by setting the <ts_mask> value to 0xfffefffe.

— T1/J1 interfaces have 24 timeslots, numbered 1 to 24. To cross connect all the timeslots on a T1 interface between the two PCM ports on a signaling board, the <ts_mask> value 0x1fffffe should be used.

In duplex mode both PCM ports should have been previously configured under the same type of PCM connector T1, E1 or J1.

<pattern>

This parameter should be set to zero.

8.2 Maintenance Module Commands

The maintenance module commands are:

- MGMT_MOD_ID, MAINT_MOD_ID & TRACE_MOD_ID Commands

8.2.1 MGMT_MOD_ID, MAINT_MOD_ID & TRACE_MOD_ID Commands

Synopsis

These commands are used to modify the default module_ids used by the s7_mgt utility to configure the Management ID, Maintenance ID and Trace ID for Protocol modules; this permits the user to specify the separate destinations to be used for trace, maintenance and management messages.

Syntax

MGMT_MOD_ID <mgmt_id>
MAINT_MOD_ID <maint_id>
TRACE_MOD_ID <trace_id>

Examples

MGMT_MOD_ID 0xcf
MAINT_MOD_ID 0xdf
TRACE_MOD_ID 0xef

Parameters

<mgmt_id>, <maint_id>, <trace_id>

The user may specify the module_ids to use for mgmt_id, maint_id and trace_id respectively. The table below shows how these apply to specific protocols.

If the command is not used then the module_ids take the default value 0xef.

Protocol	Management ID	Maintenance ID	Trace ID
MTP2	mgmt_id	-	trace_id
Q.SAAL	mgmt_id	-	trace_id
MTP3	mgmt_id	-	trace_id
ISUP	ISUP user id	ISUP user id	trace_id
TUP	TUP user id	TUP user id	-
SCCP	mgmt_id	maint_id	trace_id
TCAP	mgmt_id	maint_id	trace_id
MAP	mgmt_id	maint_id	trace_id
INAP	mgmt_id	maint_id	trace_id
IS41	mgmt_id	maint_id	trace_id
SCTP/SCTPD	mgmt_id	-	trace_id
M2PA	mgmt_id	-	trace_id

Protocol	Management ID	Maintenance ID	Trace ID
M3UA	mgmt_id	maint_id	trace_id
SUA	mgmt_id	-	trace_id

8.3 Monitor Configuration Commands

The monitor configuration commands are:

- MONITOR LINK Command (for HSL/LSL Links)
- MONITOR LINK Command (for ATM Links)

8.3.1 MONITOR LINK Command (for HSL/LSL Links)

Synopsis

This command is used to configure a signaling link to operate in receive only mode so that received signaling messages are passed directly to the user application without further processing.

Note: For the SPCI boards the ss7.dc3 code file does not support the use of the MONITOR_LINK command. When using the SPCI board for monitoring applications users must select the mon.dc3 code file.

Note: Often, applications that use MONITOR_LINK also require the line interfaces to operate in high impedance or Protected Monitoring Point mode. When using SS7HD, SS7MD or SS7LD boards high impedance and PMP modes can be selected for a particular LIU using the <liu_type> parameter in the LIU_CONFIG command.

Syntax

MONITOR_LINK <link_id> <board_id> <blink> <stream> <timeslot> <user_module> <reserved1> <flags> <reserved2> [<data_rate>]

Example

MONITOR_LINK 0 0 0-0 0 16 0x0d 0 0 0x00

Parameters

<reserved1>, <reserved2>

These parameters are reserved for future use and should be set to zero.

<link_id>

The unique logical identity of the link. It must be in the range 0 to one less than the total number of signaling links supported.

<board_id>

The ID of the board that will process the incoming signaling.

<blink>

For SPCI, SS7MD and SS7LD Boards

This is the index of the signaling link. It must be in the range 0 to one less than the number of signaling links licensed on the board.

For SS7HD boards

This is a compound parameter that indicates the signaling processor and the channel on the signaling processor that will be monitored. It is represented in the form sp_id - sp_channel where:

— sp_id is the identifier of the signaling processor with a value in the range 0 to one less than the number of processors on the board.

— sp_channel is the identifier of the channel on the signaling processor with a value in the range 0 to one less than the number of links supported per signaling processor.

The MONITOR_LINK and MTP_LINK commands cannot be used on the same sp_id/sp_channel resource.

For HSL operation, only one link per signaling processor is supported so sp_channel must be set to 0.

<stream>

When the <timeslot> parameter is set to a non-zero value, the <stream> parameter is the logical identity of the T1/E1 LIU (liu_id) containing the signaling link. It should be in the range 0 to one less than the number of LIUs.

Set both the <stream> and <timeslot> parameters to 0 to disable automatic configuration. The signaling path should be set up manually using the switch control messages.

<timeslot>

The timeslot used for signaling in the range 0-31. The valid range for an E1 interface is 1 to 31 and for a T1 interface 1 to 24.

To disable automatic configuration both <stream> and <timeslot> should be set to zero. The signaling path should then be set up manually using the switch control messages.

For HSL operation <timeslot> should be set to 0xff and the Data rate is set using the optional data rate parameter, if not present data rate defaults based on LIU type (T1/E1).

<user_module>

The module ID of the process that will receive the incoming signaling messages, passed as API_MSG_RX_IND messages.

<flags>

Per-link flags for monitoring operation.

— Bit 0 - Reserved, should be set to zero.

— Bit 1 - Enable Fill In Signal Units (FISUs) monitoring.

— Bit 10 - Set to the same value as in the MTP_LINK command.

— Bit 11 - Set to the same value as in the MTP_LINK command.

— Bit 12 - Set to the same value as in the MTP_LINK command.

— All other bits should be set to 0.

<data_rate>

An optional parameter used for SS7HD and SS7MD boards as follows:

For SS7HD boards used to specify the HSL format as follows:

Value	Description
E1_HSL	unstructured E1 HSL operation
T1_HSL	unstructured T1 HSL operation
E1_FRAMED	structured 31 slot E1 operation
T1_FRAMED	structured 24 slot T1 operations
E1_PCM	structured 30 slot E1 operation (where timeslots 0 and 16 are not used for signaling)

For SS7MD boards used to differentiate between signaling formats as follows:

Value	Description
TDM	single timeslot SS7 LSL (default)
E1_FRAMED	HSL structured 31 slot E1 operation
T1_FRAMED	HSL structured 24 slot T1/J1 operations
E1_PCM	HSL structured 30 slot E1 operation (where timeslots 0 and 16 are not used for signaling)
ATM	The command follows the syntax for ATM links

8.3.2 MONITOR LINK Command (for ATM Links)

Synopsis

This command is used user to configure an ATM link to operate in receive only mode for monitoring purposes. This functionality is only supported on the SS7MD board. The command is differentiated based on the data rate parameter. Received signaling messages are passed directly to a user application without further processing. If an ATM link is specified, multiple MONITOR_LINK commands may reference the same ATM cell stream provided the cell stream VPI-VCI combination is unique.

Syntax

MONITOR_LINK <link_id> <board_id> <blink> <atm_stream> <vpi-vci> <user_module> <reserved1> <flags> <reserved2> [<data_rate>]

Example

MONITOR_LINK 0 0 0 0 8-100 0x0d 0 0x0000 0x06 ATM

Parameters

<reserved1>, reserved2>

These parameters are reserved for future use and should be set to zero.

<board_id>

The logical identity of the board in the range from 0 to one less than the number of boards supported. This must be the same value as used in the ATM_STREAM command.

<blink>

The index of the signaling link. It must be in the range 0 to one less than the number of signaling links licensed on the board.

<atm_stream>

This defines the logical id of the cell stream over which the link runs.

<vpi-vci >

This is a compound parameter that identifies the vpi and vci of the ATM link to be monitored. It is represented in the form vpi-vci where:

— vpi is the Virtual Path Indicator of the signaling link within the ATM cell stream.

— vci is the Virtual Channel Indicator of the signaling link within the ATM cell stream.

8.4 MTP Configuration Commands

The MTP configuration commands are:

- MTP_CONFIG Command
- MTP_LINKSET Command
- MTP_LINK Command (for HSL/LSL Links)
- MTP_LINK Command (for ATM Links)
- MTP_ROUTE Command
- MTP_USER_PART Command
- MTP_TRACE Command

8.4.1 MTP_CONFIG Command

Synopsis

This command sets the global configuration parameters for the Message Transfer Part (MTP).

Syntax

MTP_CONFIG <reserved1> <reserved2> <options>

Example

MTP_CONFIG 0 0 0x00040f00

Parameters

<reserved1>, <reserved2>

These parameters are reserved for backwards compatibility and should be set to zero.

<options>

A 32 bit value containing run-time options for the operation of the MTP as follows:

Bit 0 is set to 1 to disable the MTP3 message discrimination function (allowing the signaling point to receive all messages irrespective of the destination point code contained in the message) or zero to allow the discrimination function to function normally.

Bit 1 is set to 1 to disable sub-service field (SSF) discrimination. If this bit is set to zero, received MSUs whose ssf value does not match the configured ssf value for that link set are discarded.

Bit 3 is set to 1 to cause MTP3 to generate a UPU (User Part Unavailable) message to the network on receipt of a message containing a Service Indicator value that has not been configured. If set to zero the message is discarded without sending UPU.

Bit 8 is set to 1 to select ANSI operation; otherwise it must be set to zero.

Bits 9 and 20 are used to select the point codes used in the MTP routing label as defined below:

Bit 9	Bit 20	Point Code	Description
0	0	14-bit	ITU
0	1	16-bit	Japan
1	0	24-bit	ANSI

Bit 10 is set to 1 for ANSI operation; otherwise it is set to zero.

Bit 11 is set to 1 for ANSI operation; otherwise it is set to zero.

Bit 18 is used to control MTP functionality in the event of detection of RPO (Remote Processor Outage). If set to 1, RPO is handled in accordance with the ITU-T 1992 (and later) recommendations. If set to zero, on detection of RPO the signaling link is taken out of service and restoration commenced. This bit is usually set to 1.

Bit 20 used in conjunction with bit 9 to select point codes (see above).

Bit 21 is set to 1 for use in Japanese networks; otherwise it must be set to zero.

All other bits are reserved for future use and must be set to zero.

Note: For correct ANSI operation bits 8, 9, 10, 11, and 18 must all be set to 1. This gives a typical<options> field value of 0x00040f00.

8.4.2 MTP_LINKSET Command

Synopsis

This command configures a link set to an adjacent signaling point.

Syntax

MTP_LINKSET <linkset_id> <adjacent_spc> <num_links> <flags> <local_spc> <ssf>

Example

MTP_LINKSET 0 321 2 0x0000 456 0x8

Parameters

<linkset_id>

The logical identity of the link set, in the range 0 to one less than the number of link sets supported, The linkset_id is used in other commands for reference.

<adjacent_spc>

The signaling point code of the adjacent signaling point.

<num_links>

The number of links to be allocated to the link set.

<flags>

A 16 bit value containing run-time options for the link set as follows:

Bit	Meaning
0	This bit is used to determine whether or not the user has supplied a per-link set local point code for this link set. If not, the point_code parameter from the global configuration message is used instead. • 0 - Use the per module (default) point_code as the local point code • 1 - Use the local_pc parameter as the local point code for this link set
1	This bit is used to determine whether or not the user has supplied a per-link set subservice-field (SSF) for this link set. If not, the ssf parameter from the global configuration message is used instead. • 0 - Use the per-module (default) SSF for this link set • 1 - Use the per-link set ssf parameter for this link set
2	This bit must be set to 1 when the message is being used to modify the existing link set configuration. • 0 – Normal setting; used when link set is first configured • 1 – Reconfiguration; used when the link set is being modified
3	This bit is used to enable restart procedures on a link set. • 0 - Normal setting • 1 - Restart procedures enabled. Note: Use of MTP Restart is recommended for all link sets including the inter-chassis link set on a dual system.
4	This bit is used to enable SNMP indications for this link set • 0 - SNMP disabled • 1 - SNMP enabled
11	This bit is used to automatically activate the links in this linkset • 0 – Auto-activate disabled • 1 – Auto-activate enabled
15	This bit is used to indicate that the link set is the inter-MTP3 link set connecting together the two halves when operating in a dual MTP3 configuration. • 0 – Normal setting • 1 – This link set is the inter-MTP3 link set in a dual configuration
Other bits	All other bits are reserved for future use and must be set to zero.

<local_spc>

The signaling point code of the signaling point itself.

<ssf>

The value to be used in the sub-service field of all level 3 messages and checked for by the discrimination function in all received messages. This is a 4 bit value.

Note: For ANSI operation both of the two least significant bits must be set to 1.

Note: For correct operation the adjacent point code must also have its own MTP_ROUTE declaration.

8.4.3 MTP_LINK Command (for HSL/LSL Links)

Synopsis

This section describes the MTP_LINK command format used to configure an MTP signaling link for Low Speed Link (LSL) or High Speed Link (HSL) operation. All boards support LSL operation but HSL operation is only supported on SS7HD and SS7MD boards.

Syntax

MTP_LINK <link_id> <linkset_id> <link_ref> <slc> <board_id> <blink> <stream> <timeslot> <flags> [<data rate>]

Examples

For SPCI, SS7MD and SS7LD: **MTP_LINK 0 0 2 2 0 1 0 16 0x0006 TDM**

For SS7HD: **MTP_LINK 0 0 2 2 0 1-4 0 16 0x0006 TDM**

Parameters

<link_id>

The link's unique logical link identity. It must be in the range 0 to one less than the total number of signaling links supported.

<linkset_id>

The logical identity of the link set to which the link belongs. The linkset must already have been configured using the MTP_LINKSET command.

<link_ref>

The logical identity within the link set of the signaling link. It must be in the range 0 to one less than the number of links in the link set.

<slc>

The signaling link code for the signaling link. This must be unique within the link set and is usually the same as the <link_ref>. The valid range is 0 to 15.

<board_id>

The board id of the signaling processor allocated for this signaling link.

<blink>

For SPCI, SS7MD and SS7LD Boards

This is the index of the signaling link. It must be in the range 0 to one less than the number of signaling links licensed on the board.

For SS7HD boards

This is a compound parameter that indicates the signaling processor and the channel on the signaling processor that will be monitored. It is represented in the form sp_id - sp_channel where:

— sp_id is the identifier of the signaling processor with a value in the range 0 to one less than the number of processors on the board.

— sp_channel is the identifier of the channel on the signaling processor with a value in the range 0 to one less than the number of links supported per signaling processor.

The MONITOR_LINK and MTP_LINK commands cannot be used on the same sp_id/sp_channel resource.

For HSL operation, only one link per signaling processor is supported. Therefore sp_channel must be 0.

<stream>

When the **<timeslot>** parameter is set to a non-zero value, the **<stream>** parameter is the logical identity of the T1/E1 line interface (liu_id) containing the signaling link It must be in the range 0 to one less than the number of line interfaces.

Note: For SPCI2S. Stream identifiers for the PCM interfaces are implemented on streams 2 and 3.

Note: For SS7HD. If set to 0x90, 0x91, 0x92, or 0x93, depending on the number of signaling processors, specifies the use of a specific signaling processor. In these cases, the timeslot should be the signaling processor's signaling link in the range 0 to 31.

<timeslot>

The timeslot used for signaling in the range 1 ... 31. For an E1 interface the valid range is 1 ... 31. For a T1 interface the valid range is 1 ... 24. When set to zero the signaling path through the board must be set up manually using the switch control messages.

For HSL operation <timeslot> should be set to 0xff and the Data rate is set using the optional data rate parameter, if not present data rate defaults based on LIU type (T1/E1).

<flags>

A 32 bit value containing additional run-time options.

Bit 0 is set to 1 to force the use of the emergency proving period during link alignment or zero to use the appropriate proving period according to the MTP3 recommendations.

Bit 1 is set to 1 to cause a signaling link test (in accordance with ITU-T Q.707 / ANSI T1.111.7) to be carried out before a link is put into service, or zero if a test is not required.

Bit 2 is set to 1 to cause a signaling link test (in accordance with ITU-T Q.707 / ANSI T1.111.7) to be carried out every 30 seconds. This bit is ignored unless bit 1 is also set to 1.

Bit 8 is used to select the MTP2 error correction mode. It is set to 1 to select PCR (Preventive Cyclic Retransmission) operation or zero for the Basic Method of Error Correction.

Bits 10 and 11 together select the appropriate operating bit rate for the link. The table below specifies the appropriate values for 48, 56 or 64 kb/s.

Bit 11	Bit 10	Data Rate
0	0	64 kb/s
0	1	48 kb/s[1]
1	1	56 kb/s[1]

Notes:

1. When using a serial port (SPCI2S only), 48kbit/s or 56kbit/s operation is only supported when the clock is applied externally.

2. For unstructured HSL operation (SS7HD only), these bits should be set to 0.

3. For framed HSL operation (SS7HD and SS7MD), these bits select the bit rate for each slot of the HSL link.

Bit 12 is used to select 12- or 7-bit sequence numbers for HSL only. This bit should be set for 12-bit sequence numbers, clear otherwise.

Bit 13 is only used when the link has been configured to run over a serial port. If set to 1 an external clock is used (Receive clock). If set to zero an internal clock (Transmit clock) is used. If the link has not been configured to run over a serial port, this bit must be set to zero. This bit is only applicable for SPCI2S boards and should otherwise be set to zero.

Bit 14 is set to 1 to use a serial port rather than a PCM timeslot for this link. In this mode the stream and timeslot parameters for this link are ignored (and must be set to zero). If this bit is set to zero, the link uses the specified stream and timeslot. This bit is only applicable for SPCI2S boards and should otherwise be set to zero. The serial port used by the signaling processors for each link is fixed, according to the following table:

Blink	Serial Port
0	B
1	A
2	Cannot be used for a serial port.
3	Cannot be used for a serial port.

Bit 15 is set to 1 to disable the link or zero to enable the link.

Bit 30 is set to 1 to enable SNMP indications for individual MTP links.

Bit 31 is set to 1 to denote the link as using the M2PA protocol. If selected, then blink identifies the Snlink to be used. Board_id, timeslot and stream should be set to 0.

All other bits are reserved for future use and must be set to zero.

<data_rate>

An optional parameter to specify link parameters, required for HSL or ATM operation. The valid values are:

Value	Description	Support			
		SPCI	SS7HD	SS7MD	SS7LD
TDM	single timeslot SS7 LSL (default)	•	•	•	•
E1_FRAMED	HSL structured 31 slot E1 operation		•	•	
T1_FRAMED	HSL structured 24 slot T1/J1 operation		•	•	
E1_PCM	HSL structured 30 slot E1 operation (where timeslots 0 and 16 are not used for signaling)		•	•	
ATM	The command follows the syntax for ATM links			•	

8.4.4 MTP_LINK Command (for ATM Links)

Synopsis

This command configures an ATM signaling link. ATM operation is only supported on the SS7MD board.

Syntax

MTP_LINK <link_id> <linkset_id> <link_ref> <slc> <board_id> <blink> <atm_stream> <vpi-vci> <flags> ATM

Example

MTP_LINK 0 0 0 0 3 0 0 8-100 0x0006 ATM

Parameters

The parameters **<link_id>**, **<linkset_id>**, **<link_ref>** and **<slc>** are common to the MTP_LINK command for HSL/LSL links (refer to section 8.4.3).

<board_id>

The logical identity of the board in the range from 0 to one less than the number of boards supported. This should be the same value as used in the ATM_STREAM command. If the value selected is different, then the configuration will be rejected.

<blink>

The index of the signaling link. It must be in the range 0 to one less than the number of signaling links licensed on the board.

<atm_stream>

This defines the logical id of the cell stream over which the link runs. It must be in the range 0 to one less than the combined number of ATM Cell Streams supported by all the SS7MD boards in the system.

<vpi-vci >

This is a compound parameter that identifies the vpi and vci of the ATM link. It is represented in the form vpi-vci where:

— vpi is the Virtual Path Indicator of the signaling link within the ATM cell stream.

— vci is the Virtual Channel Indicator of the signaling link within the ATM cell stream.

Note: Users should normally select a vpi/vci combination, with vpi in the range 0 to 15 and a vci in the range 0-511 (0, 3 and 4 are reserved). The vpi/vci combination associated with the link must not be the same as the default vpi/vci combination on the underlying cell stream and must be unique within the cell stream.

<flags>

A 32 bit value reserved for future use and must be set to zero.

8.4.5 MTP_ROUTE Command

Synopsis

This command configures the route to a remote point code.

Syntax

MTP_ROUTE <dpc> <norm_ls> <user_part_mask> <flags> <second_ls>

Example

MTP_ROUTE 567 0 0x0020 0x0000 0

Parameters

<dpc>

The point code of the remote signaling point for which this command is configuring routing data. It may be either an adjacent point code or a point code accessible via an adjacent Signaling Transfer Point (STP).

<norm_ls>

The linkset_id of the normal link set used to reach the specified destination. The norm_ls must be a linkset_id that has already been configured using the MTP_LINKSET command. The normal link set may be any of the following:

- The only link set used to reach the destination.
- The preferred link set used to reach the destination.
- One of a pair of links sets forming a combined link set.

In the latter two cases a second link set **(second_ls)** must also be specified.

Within a link set messages are automatically load-shared across links using the Signaling Link Selection (SLS) field in the message.

<second_ls>

The linkset_id of an optional second link set used to reach the specified destination. This may be either of the following options:

- The secondary link set used to reach the destination only on failure of the preferred link set.
- One of a pair of links sets forming a combined link set over which load-sharing takes place. (in this case bit 1 must also be set in the **<flags>** parameter of the command).

When a second link set is specified the user must also set **bit 0** in the **<flags>** field of this command.

<user_part_mask>

This is a 16 bit field used to identify the user parts that are supported over this route. The bits are labelled 0 to 15 and for each user part supported the bit corresponding to the Service Indicator for that user part must be set. (e.g., To support just ISUP messages, the ISUP Service Indicator is 5 so bit 5 should be set. Therefore a value of 0x0020 would be appropriate).

<flags>

A 16 bit field containing run-time configuration options for the route as follows:

Bit 0 is set to 1 to indicate that a second link set is specified within the command. If zero the second_ls parameter is ignored.

Bit 1 is used to determine whether or not to load-share messages across the two link sets. It is only used when two link sets are specified for the route. When set the MTP3 module load-shares messages for the destination equally across each of the two specified link sets. Otherwise the MTP3 module considers the normal link set to be the preferred link set and only uses the second link set in the event of failure of the normal link set. The bit may be set to 1 to enable load-sharing across the two link sets, or zero to disable load-sharing and use preferred and secondary link sets.

Bit 2 is used to indicate this is a default route permitted to carry traffic for any unknown DPC

Bit 3 is used to enable Pseudo DPC operation - used in conjunction with bit 2 to control the behavior of default routes. When set the route is considered available to carry traffic as soon as either link set is accessible. MTP3 does not generate Route Set Test messages or expect Transfer Allowed messages for this "default" route

Bit 4 is used to enable timer T103 – buffering of messages for up to 10 seconds in the event that the destination becomes inaccessible, allowing for recovery of the route.

Bit 5 is used to disable route-set-test for this route

Bit 6 is used to activate SNMP indications for the route

All other bits are reserved for future use and must be set to zero.

8.4.6 MTP_USER_PART Command

Synopsis

This command is used to configure a local user part module in situations when the user part does not already have its own configuration in the config.txt protocol configuration file.

Syntax

MTP_USER_PART [NC] <si> <module_id>

Example

```
MTP_USER_PART       0x03  0x2d
MTP_USER_PART  NC1  0x05  0x3d
```

Parameters

[NC]
Optional Network Context parameter (if not present defaults to NC0).
The use of the NC parameter is only required for M3UA Multiple LAS configurations, where NC0 corresponds to LAS1, NC1 corresponds to LAS2 and so on.

<si>
The service indicator.

<module_id>
The module id of the user process.

Note: This command must not be used when the corresponding configuration commands are used (ISUP_CONFIG, TUP_CONFIG, SCCP_CONFIG, etc) as these commands automatically perform the function of the MTP_USER_PART command for the default NC (NC0).

8.4.7 MTP_TRACE Command

Synopsis

This command sets the MTP3 trace masks.

Syntax

MTP_TRACE <op_evt_mask> <ip_evt_mask>

Example

MTP_TRACE 0x0001 0x0001

Parameters

<op_evt_mask>
Output event trace mask. For full description of use refer to the *MTP3 Programmer's Manual*.

<ip_evt_mask>
Input event trace mask. For full description of use refer to the *MTP3 Programmer's Manual*.

8.5 ATM Configuration Commands

ATM configuration is only supported on the SS7MD board. The ATM configuration commands are:

- ATM_CONFIG Command
- ATM_STREAM Command (Configure ATM Cell Stream)
- ATM_TIMER Command (Configure Timers for Q.SAAL Links)

8.5.1 ATM_CONFIG Command

Synopsis

Global configuration of the ATM Module.

Syntax

ATM_CONFIG <options> <num_streams>

Example

ATM_CONFIG 0x0000 4

Parameters

<options>
A 16-bit value containing additional run-time options. The bit significance is as follows:

— Bit 0 - Use ATM Forum Idle cell format rather than ITU.

<num_streams>
The maximum number of cell streams per board this module will be asked to simultaneously support. For an IMA bundle, each TDM stream within the bundle is counted separately.

8.5.2 ATM_STREAM Command (Configure ATM Cell Stream)

Synopsis

Configures an ATM Cell Stream.

Syntax

ATM_STREAM <id> <board_id> <cellstream_id> <liu_id> <options> <ima_frame_len> <max_frame_len> <def_vpi> <def_vci> <timeslot>

Example

ATM_STREAM 3 0 3 3 0x06 0 280 12 10 0xfffefffe

Parameters

<id>

The logical Cell Stream ID from the user's perspective.

<board_id>

The board ID of the signaling processor allocated for this ATM link.

<cellstream_id>

The Layer 2 ID of the cell stream within the board. In the range of 0 to one less than the number of cell streams supported per board.

<liu_id>

Line Interface Units (LIUs) to be used by the cell stream. If IMA is active (Bit 3 of the <options> parameter), the parameter is a bitmap of the LIUs to be used by the bundle (bit 0 = LIU 0, etc.). If IMA is not active, the parameter identifies the LIU to be used.

<options>

A 16-bit value containing additional options for the ATM link. The bit significance is as follows:

— Bit 0 - Enable payload scrambling

— Bit 1 - Use ATM coset in HEC calculation. When terminating Q.SAAL links on the cell stream this bit must be set. When monitoring links values of 0 or 1 are permitted.

— Bit 2 - Autocorrect invalid cells if possible

— Bit 3 - Configuration describes an IMA bundle

Note: Either Payload Scrambling or ATM Coset mode, or both, must be enabled for correct operation.

<ima_frame_len>

The length of the IMA frame (for IMA use only).

Value	Options
1	32 cells per IMA frame
2	64 cells per IMA frame
3	128 cells per IMA frame
4	256 cells per IMA frame

Note: For non IMA streams this field is reserved and should be set to zero.

<max_frame_len>

The maximum length of a reassembled (AAL) frame. Frames longer than this will be discarded by the ATM layer. Recommended value is 280.

<def_vpi>

A default AAL5 link will be configured for the cell stream to signal incoming active connections. This is the VPI that will be used for this connection.

<def_vci>

A default AAL5 link will be configured for the cell stream to signal incoming active connections. This is the VCI that will be used for this connection. Values 0, 3, and 4 are reserved and should not be used.

Note: The default VPI/VCI combination configured here must not be specified for any AAL5 link on this cell stream.

<timeslot>

Bitmap of active timeslots within the above TDM streams. Typically the timeslot bitmap for E1 will be 0xfffefffe and for T1/J1 will be 0x01fffffe.

8.5.3 ATM_TIMER Command (Configure Timers for Q.SAAL Links)

Synopsis

This command allows specific timer values to be set for STM links. Otherwise default values are used.

Syntax

ATM_TIMER <reserved> <timer_id> <value>

Example

ATM_TIMER 0 T1 10

Parameters

<reserved>

This parameter is reserved for future use and should be set to zero.

<timer_id>

The identifier of the timer to be changed. It should be set to one of the following tokens: CC, KEEP_ALIVE, NO_RESP, POLL, IDLE, T1, T2, T3.

<value>

The timer value in milliseconds. Any timers not explicitly configured use the default values shown.

Timer ID	Default Value (ms)	Range (min - max)
CC	1,500	15 - 2,500
KEEP_ALIVE	300	15 - 2,500
NO_RESP	1,500	100 - 10,000
POLL	100	20 - 600
IDLE	100	20 - 600
T1	5,000	1,000 - 20,000
T2	120,000	10,000 - 300,000
T3	10	1 - 30

8.6 ISUP Configuration Commands

The ISUP configuration commands are:

- ISUP_CONFIG Command
- ISUP_CFG_CCTGRP Command (Circuit Group Configuration)
- ISUP_TIMER Command (ISUP Timer Configuration)

8.6.1 ISUP_CONFIG Command

Synopsis

This command sets the global configuration parameters for the ISUP module.

Syntax

ISUP_CONFIG <res1> <res2> <user_id> <options> <num_grps> <num_ccts> [<partner_id>]

Example

ISUP_CONFIG 0 0 0x2d 0x0435 4 128

Parameters

<res1>, <res2>

Reserved for backwards compatibility. These fields should be set to zero.

<user_id>

The module_id of the application running on the host that uses the ISUP module.

<options>

A 16 bit value containing global run-time options for the operation of the ISUP module. The meaning of each bit is as defined for the 'options' parameter in the ISUP Configure Request message as detailed in the *ISUP Programmer's Manual*

<num_grps>

The maximum number of ISUP circuit groups that the user intends to use. This must not exceed the maximum number of circuit groups supported otherwise module configuration will fail. Typically **<num_grps>** would be set to the maximum number of circuit groups supported.

<num_ccts>

The maximum number of ISUP circuits that the user intends to use. This must not exceed the maximum number of circuits supported otherwise module configuration will fail. Typically **<num_ccts>** is set to 32 times the number of groups for E1 operation and 24 times the number of circuit groups for T1 operation.

Note: The valid range for the circuit identifier (cid) is from zero up to one less than the maximum number of circuits (<num_ccts>).

<partner_id>

Optional parameter for use when operating in dual resilient configuration. This parameter is the module_id of the 'partner' ISUP module (equivalent to the 'module_id field in the ISUP Configure Request message as documented in the ISUP Programmer's Manual).

8.6.2 ISUP_CFG_CCTGRP Command (Circuit Group Configuration)

Synopsis

This command sets the configuration parameters for a group of ISUP circuits. Usually a group is all the circuits on a single E1 or T1 interface.

Syntax

ISUP_CFG_CCTGRP <gid> <dpc> <base_cic> <base_cid> <cic_mask> <options> <user_inst> <user_id> <opc> <ssf> <variant> <options2>

Example

ISUP_CFG_CCTGRP 0 3 1 1 0x7fff7fff 0x00000003 0 0x2d 2 0x8 4 0x00000000

Parameters

<gid >

The group id of the circuit group in the range 0 to one less than the number of groups supported.

<dpc>

The destination point code for all circuits in the circuit group.

<base_cic>

The Circuit Identification Code (CIC) that is allocated to the first circuit in the circuit group.

<base_cid>

The logical id for the first circuit in the circuit group. It must lie in the range 0 to one less than the number of circuits supported.

<cic_mask>

A 32 bit mask with bits set to indicate which circuits are to be allocated to the circuit group. Bit zero must always be set as it represents the base_cic/base_cid. Subsequent bits represent the subsequent circuits.

Note: ANSI circuit groups are not permitted to contain more than 24 circuits.

<options>

A 32 bit value containing run-time options for the ISUP circuit group (see "Configure Circuit Group Request" section of the *ISUP Programmer's Manual*). Bits 0 through 15 are equivalent to the "options" field and bits 16 through 31 represent the "ext_options" field as detailed in the ISUP Programmer's Manual.

<user_inst>

The instance number of the user application. Typically only a single user application exists so this field would be set to zero.

<user_id>

The module_id of the user application.

<opc>

Originating Point Code. The local point code for all circuits in the group.

<ssf>

The value to be used in the sub-service field of all ISUP messages for this circuit group.

<variant>

The protocol "variant" for this circuit group. Refer to the ISUP Programmer's Manual for full details.

<options2>

A 32 bit value containing additional run-time options for the ISUP circuit group (see "Configure Circuit Group Request" section of the *ISUP Programmer's Manual*). Bits 0 through 31 are equivalent to the "ext_1_options" as detailed in the *ISUP Programmer's Manual*.

8.6.3 ISUP_TIMER Command (ISUP Timer Configuration)

Synopsis

This command provides the ability to configure the ISUP protocol timers from the config.txt file.

Syntax

ISUP_TIMER <reserved> <timer_id> <value>

Example

ISUP_TIMER 0 T4 550

Parameters

<reserved>

Must be set to 0. Reserved for future use.

<timer_id>

The text identifier for the timer to be configured as shown below in Table 9.

<value>

The timer value in seconds, except T29 and T30 which are in multiples of tenths of a second (100 ms). Any timers not explicitly set are set to their default values, as shown below in Table 9

Table 9. ISUP Default Timer Values

Timer Mnemonic	Default Value (Seconds)	Timer Mnemonic	Default Value (Seconds)	Timer Mnemonic	Default Value (Seconds)
T1	10	T15	60	T28	10
T2	180	T16	10	T29	0.5
T3	180	T17	60	T30	8
T4	300	T18	10	T33	14
T5	60	T19	60	T34	3
T6	180	T20	10	T35	20
T7	25	T21	60	T36	13
T8	13	T22	10	T38	150
T9	45	T23	60	T39	10
T10	5	T24	2	T103	20
T12	10	T25	5	T104	3
T13	60	T26	120		
T14	10	T27	240		

8.7 TUP Configuration Commands

The TUP configuration commands are:

- TUP_CONFIG Command (Global TUP Configuration)
- TUP_CFG_CCTGRP Command (Circuit Group Configuration)

8.7.1 TUP_CONFIG Command (Global TUP Configuration)

Synopsis

This command sets the global configuration parameters for the TUP module.

Syntax

TUP_CONFIG <res1> <res2> <user_id> <options> <num_grps> <num_ccts> <partner_id>

Example

TUP_CONFIG 0 0 0x2d 0x8541 4 128

Parameters

<res1>, <res2>

Reserved for backwards compatibility. These fields should be set to zero.

<user_id>

The module_id of the application running on the host that uses the TUP module.

<options>

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 Configure Request message as detailed in the *TUP Programmer's Manual*.

<num_grps>

The maximum number of TUP circuit groups that the user intends to use. This must not exceed the maximum number of circuit groups supported otherwise module configuration will fail. Typically <num_grps> would be set to the maximum number of circuit groups supported.

<num_ccts>

The maximum number of TUP circuits that the user intends to use. This must not exceed the maximum number of circuits supported otherwise module configuration will fail. Typically <num_ccts> is set to 32 times the number of groups for E1 operation and 24 times the number of circuit groups for T1 operation.

Note: The valid range for the circuit identifier (cid) is from zero up to one less than the maximum number of circuits (num_ccts).

<partner_id>

Optional parameter for use when operating in dual resilient configuration. This parameter is the module_id of the "partner" TUP module (equivalent to the "ucic_id" field in the TUP Configure Request message as documented in the *TUP Programmer's Manual*).

8.7.2 TUP_CFG_CCTGRP Command (Circuit Group Configuration)

Synopsis

This command sets the configuration parameters for a group of TUP circuits. Usually a group is all the circuits on a single E1 or T1 interface.

Syntax

TUP_CFG_CCTGRP <gid> <dpc> <base_cic> <base_cid> <cic_mask> <options> <user_inst> <user_id> <opc> <ssf> <variant> <options2>

Example

TUP_CFG_CCTGRP 0 3 1 1 0x7fff7fff 0x00000003 0 0x2d 123 0x8 0 0x0

Parameters

<gid >

The group id of the circuit group in the range 0 to one less than the number of groups supported.

<dpc>

The destination point code for the circuits in the circuit group.

<base_cic>

The Circuit Identification Code (CIC) that is allocated to the first circuit in the circuit group.

<base_cid>

The logical id for the first circuit in the circuit group. It must lie in the range 0 to one less than the number of circuits supported.

<cic_mask>

A 32 bit mask with bits set to indicate which circuits are to be allocated to the circuit group. Bit zero must always be set as it represents the base_cic/base_cid. Subsequent bits represent the subsequent circuits.

<options>

A 32 bit value containing run-time options for the TUP circuit group (see "Configure Circuit Group Request" section of the TUP Programmers Manual).

<user_inst>

The instance number of the user application. Typically only a single user application exists so this field would be set to zero.

<user_id>
The module_id of the user application.

<opc>
Originating Point Code. The local point code for all circuits in the group.

<ssf>
The value to be used in the sub-service field of all TUP messages for this circuit group.

<variant>
This field is reserved for future use and must be set to zero.

<options2>
This field is reserved for future use and must be set to zero.

8.8 SCCP Configuration Commands

The SCCP configuration commands are:

- SCCP_CONFIG Command
- SCCP_SSR Command (Configure SCCP Sub-System Resource)
- SCCP_CONC_SSR Command (Configure Concerned SSR)
- SCCP_TRACE Command
- SCCP_GTT_PATTERN Command (Define Global Title Pattern)
- SCCP_GTT_ADDRESS Command (Define Global Title Address)
- SCCP_GTT Command (Add Entry in GTT Table)

8.8.1 SCCP_CONFIG Command

Synopsis

The SCCP_CONFIG command supplies the global configuration parameters for the SCCP protocol, activating the SCCP and TCAP protocols.

Syntax

SCCP_CONFIG <local_spc> <ssf> <options> [<options2> [<partner_id> <instance> <smb_flags>]]

Example

SCCP_CONFIG 123 8 0

Parameters

<local_spc>
The local point code.

<ssf>
The sub-service field value that SCCP uses when exchanging messages with the MTP. This value must always be set so that the Network Indicator bits (the two most significant bits of the 4-bit ssf value) match those set in the MTP_LINKSET command.

<options>
A 32-bit value containing run-time configuration options for the SCCP module. The 16 least significant bits provide the 'options' parameter and the 16 most significant bits provide the 'ext_options' parameter, as defined in the *SCCP Programmer's Manual*.

Bit 0 must always be set to zero.

<options2>
Additional 32 bit run time options for the configuration and operation of SCCP. Bits 0 and 31 are used by s7_mgt during configuration as detailed below. The remaining bits map directly to the 'ext2_options' parameter as documented in the Module Configuration Request section of the *SCCP Programmer's Manual*.

Bit 0 Send_User In Service (UIS). When set to 1 this bit causes s7_mgt to automatically generate and send UIS messages to SCCP for all configured local sub-systems. By default the bit is 0 and the user application is responsible for generating UIS messages.

Bit 31 is used to activate SCCP Connection Oriented operation. When set to zero s7_mgt configures SCCP for Connectionless operation. When set to 1 s7_mgt configures SCCP for Connection Orientated operation using the fixed configuration parameter values shown below. Note that for SCCP operation two license types are available: SCCP-CL which only permits connectionless operation and SCCP-CO which supports both connectionless and connection oriented (Class 2 only) operation.

Parameter	Value
NUM_UC	2048
UC_ONSET	1536
UC_ABTM	1024
BASE_ID	1024
TOP_ID	2047
MIN_ID	0
MAX_ID	2047

<partner_id>

Specifies the module_id of the partner SCCP module.

<instance>

Value in the range 0 - 15 which specifies the instance of SCCP running on this system.

The <partner_id> and <instance> parameters provide the capability to configure dual chassis fault tolerant systems that appear to the network as a single point code. For further details refer to the Application Note: Enabling *Dual Chassis Fault Tolerance with Dialogic® Signaling Boards*.

<smb_flags>

Flags relating to the SCCP management broadcast mechanism. For full details refer to the Module Configuration Request section of the *SCCP Programmer's Manual*.

8.8.2 SCCP_SSR Command (Configure SCCP Sub-System Resource)

Synopsis

The SCCP_SSR command supplies the global configuration parameters for SCCP.

Syntax

SCCP_SSR <ssr_id> RSP <remote_spc> <flags> <pc_mask> [<ssf> [<mtp_module_id>]]

SCCP_SSR <ssr_id> LSS <local_ssn> <module_id> <flags> <protocol>

SCCP_SSR <ssr_id> RSS <remote_spc> <remote_ssn> <flags>

Examples

SCCP_SSR 1 RSP 1236 0

SCCP_SSR 2 LSS 0x07 0x0d 1 TCAP

SCCP_SSR 3 RSS 1236 0x67 0

Parameters

<ssr_id>
Unique ID for the SSR.

<remote_spc>
The point code of the remote signaling point, which may be either an STP or an SCP. For correct operation, <remote_spc> must always have its own RSP entry in addition to any RSS entries. There must also be an MTP_ROUTE defined for this signaling point.

<local_ssn>
The local sub-system number as defined by the SCCP protocol.

<flags>
A 16-bit value, where each bit enables or disables additional features of the RSP, RSS, or LSS. The meaning for each bit is as defined for the options parameter described in the Configure Sub-System Resource Request section of the SCCP Programmer's Manual.

<module_id>
The module identifier of the user application that implements the local sub-system.

<remote_ssn>
The remote sub-system number as defined by the SCCP protocol.

<pc_mask>
A 32-bit value specifying the part of a destination point code that must match the <remote_spc> value for a SCCP transmit message to be sent down to this destination sub-system. Bits set to 0 indicate that the corresponding bit position in the transmit message destination point code must match the bit position of the <remote_spc>, bits set to 1 indicate bit positions in the message destination point code that do not need to match the <remote_spc> set for this RSP. This allows configuration of default destination sub-systems (for example, to a gateway SCP).

<protocol>
Should be set to SCCP, TCAP, MAP, INAP or IS41 according to the layer of the protocol stack to which the user application interfaces. Note there can be at most one LSS for each of the MAP, INAP and IS41 protocols.

<ssf>
The SSF (Sub-Service Field) for use in messages sent to this RSP. If <ssf> is not present or is set to 0xff, then the default SSF value configured in the SCCP_CONFIG command will be used. The SSF value should always be configured to match the value used within MTP3 for the corresponding link set(s).

<mtp_module_id>

The mtp_module_id field allows SCCP to optionally send messages to a different MTP3 module on a per-RSP basis. If omitted or set to zero the MTP3 module_id will be used.

8.8.3 SCCP_CONC_SSR Command (Configure Concerned SSR)

Synopsis

This command marks the specified sub-system (which was declared by SCCP_SSR) as requiring notification of changes in the accessibility of another sub-system. Notification is given in the form of an SCCP management indication.

Syntax

SCCP_CONC_SSR <id> <cssr_id> <ssr_id>

Example

SCCP_CONC_SSR 1 2 3

Parameters

<id>

A unique value of local significance in the range 0 to 8191 used to identify the concerned sub-system resource.

<cssr_id>

The ID of the subsystem that will receive the notifications.

<ssr_id>

The ID of the sub-system for which state changes will be issued.

8.8.4 SCCP_TRACE Command

Synopsis

This command sets the SCCP trace masks. Refer to *SCCP Programmer's Manual* for full details.

Syntax

SCCP_TRACE <op_evt_mask> <ip_evt_mask> <non_prim_mask>

Example

SCCP_TRACE 0x1 0x1 0x1

Parameters

<op_evt_mask>

Output event trace mask.

<ip_evt_mask>

Input event trace mask.

<non_prim_mask>
Non-primitive trace mask.

8.8.5 SCCP_GTT_PATTERN Command (Define Global Title Pattern)

Synopsis

The SCCP_GTT_PATTERN command defines a global title pattern to be matched for a global title translation.

Syntax

SCCP_GTT_PATTERN <pattern_id> <addr_indicator> <pc> <ssn> <global_title> [<gtai_pattern>]

Example

SCCP_GTT_PATTERN 5 0x10 0x0000 0 0x001104 44/+

Parameters

<pattern_id>
A unique ID identifying the pattern.

<addr_indicator>
The address indicator octets, formatted according to the point-code format specified in the SCCP_CONFIG <options> parameter (see "Called Party Address", Q.713 or ANSI T1.112).

<pc>
The point code. This is ignored if bit 0 of <addr_indicator> is not set.

<ssn>
The subsystem number. This is ignored if bit 1 of <addr_indicator> is not set.

<global_title>
The global title, excluding the global title address information, specified as a string of hexadecimal octets starting with 0x.

<gtai_pattern>
The pattern of global title address information to match, specified as a string of hexadecimal digits in left-to-right order (that is, the pairs of digits are not swapped as is the case for a BCD string). In addition to hexadecimal digits, this string can contain the following characters:

Character	Function
-	Padding (ignored).
+	Wildcard - matches any number of digits. The "+" wildcard matches the shortest possible string of digits for example a pattern such as "12+67" matches "1234567", but does not match "1236767".
?	Wildcard - matches exactly one digit.
/	Separator used to split the pattern into sections. Each section can be processed differently, as specified by the <mask> parameter in the SCCP_GTT command.

8.8.6 SCCP_GTT_ADDRESS Command (Define Global Title Address)

Synopsis

This command defines a global title to be used as the primary or backup destination of a translation. The global title address information of this command is combined with the global title being translated by examining the mask provided in the SCCP_GTT command.

Syntax

SCCP_GTT_ADDRESS <address_id> <addr_indicator> <pc> <ssn> <global_title> [<gtai_replacement>]

Example

SCCP_GTT_ADDRESS 9 0x11 0x1234 0 0x001104 0-/-

Parameters

<address_id>
A unique ID identifying the address.

<addr_indicator>
The address indicator octet, formatted according to the point-code format specified in the SCCP_CONFIG <options> parameter (see "Called Party Address", Q.713 or ANSI T1.112).

<pc>
The point code. This is ignored if bit 0 of **<addr_indicator>** is not set.

<ssn>
The subsystem number. This is ignored if bit 1 of **<addr_indicator>** is not set.

<global_title>
The global title, excluding the global title address information, specified as a string of hexadecimal octets starting with 0x.

<gtai_replacement>
The global title address information to translate to, specified as a string of hexadecimal digits in left-to right order (that is, the pairs of digits are not swapped as is the case for a BCD string). In addition to hexadecimal digits, this string can contain the following characters:

Character	Function
-	Padding (ignored).
/	Separator used to split the address into sections. Each section can be processed differently, as specified by the <mask> parameter in the SCCP_GTT command.

8.8.7 SCCP_GTT Command (Add Entry in GTT Table)

Synopsis

The SCCP_GTT command adds a translation to the SCCP global title translation table. This command must be specified after the SCCP_GTT_PATTERN and SCCP_GTT_ADDRESS commands. The pattern, mask, primary and backup addresses referenced by this command must all have an identical number of sections.

Syntax

SCCP_GTT <pattern_id> <mask> <primary_address_id> [<backup_address_id>]

Example

SCCP_GTT 5 R-/K 9

Parameters

<pattern_id>

Identifies the pattern specified by the SCCP_GTT_PATTERN command. This value is also used to index the translation within the SCCP/SUA module.

<mask>

An expression detailing the operation to be applied to each section of the global title pattern. The format is exactly one operation per section and must contain exactly the same number of sections as the **<gtai_pattern>** parameter of the associated SCCP_GTT_PATTERN command and the **<gtai_replacement>** parameter of the associated SCCP_GTT_ADDRESS command. The mask can contain the following:

Mnemonic	Function
-	Padding (ignored).
/	Separator used to split the mask into sections.
K or KEEP	The digits in the corresponding section of the global title address information undergoing translation will be preserved.
R or REPLACE	The digits in the corresponding section of the global title address information undergoing translation will be replaced with digits in the corresponding section of the primary (or backup) address.

<primary_address_id>

Identifies the SCCP_GTT_ADDRESS command to use as the primary translation.

<backup_address_id>

Identifies the SCCP_GTT_ADDRESS command to use as the backup translation.

8.9 DTC Configuration Commands

The DTC configuration commands are:

- DTC_CONFIG Command
- DTC_SSR Command (Configure DTC Sub System Resource)

8.9.1 DTC_CONFIG Command

Synopsis

The DTC_CONFIG command supplies the global configuration parameters for the DTC protocol, activating DTC and higher protocols.

Syntax

DTC_CONFIG <num_servers> <server_selection> <host_number> <rsi_status_user_id>

Example

DTC_CONFIG 2 0 0 0xef

Parameters

<num_servers>

Number of servers in the system.

<server_selection>

The selection mechanism used by DTC to determine which server to be used taken from the following values:

Value	Mnemonic	Description
0	DTC_SELECT_SEQ	Selects available servers in a sequential order
1	DTC_SELECT_REV_SEQ	Selects available servers in a reverse sequential order
2 - 255		Reserved for future use.

<host_number>

The host number, which should be unique across hosts.

<rsi_status_user_id>

Module ID to forward RSI link status messages to.

8.9.2 DTC_SSR Command (Configure DTC Sub System Resource)

Synopsis

The DTC_SSR command configures a local subsystem using DTC. The command works in a similar way to the SCCP_SSR LSS command but configures the subsystem to run on top of DTC instead of SCCP. DTC and SCCP cannot be used at the same time, so the SCCP_SSR and DTC_SSR commands are incompatible with each other.

Syntax

DTC_SSR <ssr_id> LSS <local_ssn> <module_id> <reserved> <protocol>

Example

DTC_SSR 1 LSS 0x07 0x0d 0 TCAP

Parameters

<ssr_id>
A unique ID for the SSR.

<local_ssn>
The local sub-system number as defined by the SCCP protocol.

<module_id>
The module identifier of the user application on the host computer that implements the local sub-system.

<reserved>
Must be set to 0.

<protocol>
Should be set to TCAP, MAP, INAP or IS41 according to the layer of the protocol stack to which the user application interfaces.

Note: There can be at most one LSS for each of MAP, INAP and IS41.

8.10 TCAP Configuration Commands

The TCAP configuration commands are:

- TCAP_CONFIG Command
- TCAP_CFG_DGRP Command (Dialog Group Configuration)
- TCAP_TRACE Command

8.10.1 TCAP_CONFIG Command

Synopsis

The TCAP_CONFIG command provides the TCAP operating parameters and, when used, must appear after the SCCP_SSR or DTC_SSR commands. This command should only be used when an SCCP_CONFIG or DTC_CONFIG command is present. The use of the TCAP_CONFIG command is not required and TCAP is configured with default values if the TCAP_CONFIG command is not present.

By default, TCAP is configured with 32k incoming and 32k outgoing dialogs. The TCAP_CONFIG command must be used to change these parameters for systems requiring a different number of dialogs.

Syntax

TCAP_CONFIG <base_ogdlg_id> <nog_dialogues> <base_icdlg_id> <nic_dialogues> <options> <dlg_hunt> [[<addr_format>] <partner_id> <tcap_inst>[<max_instance>]]

Example

TCAP_CONFIG 0x0000 8192 0x8000 8192 0x0000 0

Parameters

<base_ogdlg_id>
The dialogue_id for the first outgoing dialog.

<nog_dialogues>
The number of outgoing dialogs to support. The valid range is 0 to 65535.

<base_icdlg_id>
The dialogue_id for the first incoming dialog.

<nic_dialogues>
The number of incoming dialogs to support. The valid range is 0 to 65535.

Note: The total number of dialogs (<nog_dialogues> + <nic_dialogues>) must not exceed 65535.

<options>
Specifies TCAP protocol options as defined for the TCAP Configuration Request message in the TCAP Programmer's Manual.

<dlg_hunt>

The hunt mode used in the case of multiple TCAP hosts to determine which TCAP group is selected whenever a new incoming dialog arrives. It should be set to 0, 1 or 2 for the following hunt modes:

Option	Function
0	Cyclic Selection. Each new incoming dialog is allocated to the next TCAP group.
1	Load Balanced Selection. Each new incoming dialog is allocated to the group with the least number of active incoming dialogs.
2	Sequential Selection. Each new incoming dialog is allocated to the group containing the first inactive incoming dialogue_id.

<addr_format>

Defines how TCAP should interpret address information from messages received from SCCP in order to direct received TCAP primitives to unique SCCP sub-systems (TCAP user applications). It should be set to 0, 1, 2, 3 or 4 for the following options:

Option	Function
0	If configured to use ITU-T PDU formats (options bit 1 not set), use the ITU-T Q.713 SCCP address format. If configured to use ANSI PDU formats (options bit 1 set), use the ANSI T1.112 SCCP address format.
1	Use the ITU-T Q.713 SCCP address format (14-bit point codes).
2	Use the ITU-T Q.713 SCCP address format modified for 24-bit point codes.
3	Use the ANSI T1.112 SCCP address format modified for 14-bit point codes.
4	Use the ANSI T1.112 SCCP address format (24-bit point codes).

<partner_id>

Specifies the module_id of the partner TCAP module.

<tcap_inst>

Value in the range 0 - 15 which specifies the instance of TCAP running on this system.

The <partner_id> and <tcap_inst> parameters provide the capability to configure dual chassis fault tolerant systems that appear to the network as a single point code. See the Application Note: Enabling Dual Chassis Fault Tolerance with Dialogic® Signaling Boards for a description of how such a configuration can be used.

<max_instance>

Specifies the maximum number of hosts considered to make up the system, in the range 0-128. When used in conjunction with an SIU using DTS this value must also be present in the DTS_CONFIG command on the SIU(s).

8.10.2 TCAP_CFG_DGRP Command (Dialog Group Configuration)

Synopsis

This command allows the user to configure TCAP dialog groups, each group handling a subset of the total available dialogs. This allows each group to reside on a separate host computer that in turn allows the application using TCAP to be distributed over several machines. If the TCAP_CFG_DGRP command is omitted, the complete range of dialog identifiers defined by the TCAP_CONFIG command is assigned.

The TCAP_CONFIG command must exist before this command in the config.txt file.

Syntax

TCAP_CFG_DGRP <gid> <base_ogdlg_id> <nog_dialogues> <base_icdlg_id> <nic_dialogues> <options> <reserved>

Example

TCAP_CFG_DGRP 0 0x0000 1024 0x8000 1024 0 0

Parameters

<gid>
A logical identifier for this group. The valid range is 0 to 31.

<base_ogdlg_id>
The first outgoing dialog ID assigned to this dialog group.

<nog_dialogues>
The number of outgoing dialogs assigned to this group, hence outgoing dialog IDs base_ogdlg_id to base_ogdlg_id + nog_dialogues-1 are assigned to this group.

<base_icdlg_id>
The first incoming dialog ID assigned to this dialog identifier group.

<nic_dialogues>
The number of incoming dialogs assigned to this group, therefore, outgoing dialog IDs base_ogdlg_id to base_icdlg_id + nic_dialogues-1 are assigned to this group.

<options>
Reserved for future use, set to 0.

<reserved>
Must be set to 0.

8.10.3 TCAP_TRACE Command

Synopsis

This command sets the TCAP trace masks. Refer to *TCAP Programmer's Manual* for full details.

Syntax

TCAP_TRACE <op_evt_mask> <ip_evt_mask> <non_prim_mask>

Example

TCAP_TRACE 0x7 0xf 0x0

Parameters

<op_evt_mask>
Output event trace mask.

<ip_evt_mask>
Input event trace mask.

<non_prim_mask>
Non-primitive trace mask.

8.11 MAP Configuration Commands

The MAP configuration commands are:

- MAP_CONFIG Command
- MAP_TRACE Command

8.11.1 MAP_CONFIG Command

Synopsis

The MAP_CONFIG command provides the MAP operating parameters and, if used, must appear after the SCCP_SSR commands in the config.txt file. The use of this command is not required and MAP is configured with default values if the MAP_CONFIG command is not present.

Syntax

MAP_CONFIG <options>

Example

MAP_CONFIG 2

Parameters

<options>
Specifies MAP protocol options as defined for the MAP Configuration Request message in the MAP Programmer's Manual.

8.11.2 MAP_TRACE Command

Synopsis

This command sets the MAP trace masks. Refer to *MAP Programmer's Manual* for full details.

Syntax

MAP_TRACE <op_evt_mask> <ip_evt_mask> <non_prim_mask>

Example

MAP_TRACE 0xf 0xf 0x4

Parameters

<op_evt_mask>
Output event trace mask.

<ip_evt_mask>
Input event trace mask.

<non_prim_mask>
Non-primitive trace mask.

8.12 INAP Configuration Commands

The INAP configuration commands are:

- INAP_CONFIG Command
- INAP_FE Command (Configure INAP Functional Entity)
- INAP_AC Command (Configure INAP Application Context)
- INAP_TRACE Command

8.12.1 INAP_CONFIG Command

Synopsis

The INAP_CONFIG command provides the INAP operating parameters and, if used, must appear after the SCCP_SSR commands in the config.txt file. The use of this command is not required and MAP is configured with default values if the INAP_CONFIG command is not present.

Syntax

INAP_CONFIG <options>

Example

INAP_CONFIG 2

Parameters

<options>
Specifies INAP protocol options as defined for the INAP Configuration Request message in the INAP Programmer's Manual.

8.12.2 INAP_FE Command (Configure INAP Functional Entity)

Synopsis

This command is used to configure the INAP functional entity records for operation. These allow the user application to refer to Functional Entities (FEs) in the network via a local reference rather than providing the full SCCP address. The user may subsequently use this reference in the "Destination FE" or "Originating FE" parameters of the INAP_OPEN_DLG primitive or in the "IN_dialogue_open" API function. This reference is used instead of the destination or origination address parameter.

Syntax

INAP_FE <fe_ref> <options> <sccp_address>

Example

INAP_FE 0x00000007 0x00000001 0x00000000

Parameters

<fe_ref>
A logical identifier for this Functional Entity (FE).

<options>
A 16-bit options value. Bit 0, when set to 1 identifies a local FE. Other bits should be set to 0.

<sccp_address>
The SCCP address of the local FE, in Q.713 format commencing with the address indicator, as a string of hex characters, up to 18 characters in length. The SIU supports up to 32 functional entities.

8.12.3 INAP_AC Command (Configure INAP Application Context)

Synopsis

This command is used to configure the INAP Application Context (AC) records for use. These control the application context negotiation that the module conducts during dialog establishment. The supported application contexts must be individually configured using this message. The module only accepts incoming dialogs with configured Application Contexts. If a dialog request with an unconfigured context is received, a dialog abort message is returned to the requesting Functional Entity. If no supported Application Contexts are configured, the application context negotiation is disabled. The module accepts all incoming dialogs.

Syntax

INAP_AC <ac_ref> <ac>

Example

INAP_AC 0x00 0xa109060704000101010000

Parameters

<ac_ref>
A logical identifier for this Application Context (AC).

<ac>
Application context. Specified as hexadecimal characters, prefixed by "0x". An application context may be up to 32 octets (character pairs) in length. The SIU supports up to 32 application contexts.

8.12.4 INAP_TRACE Command

Synopsis

This command sets the INAP trace masks. Refer to *INAP Programmer's Manual* for full details.

Syntax

INAP_TRACE <op_evt_mask> <ip_evt_mask> <non_prim_mask>

Example

INAP_TRACE 0xf 0xf 0x7f

Parameters

<op_evt_mask>
Output event trace mask.

<ip_evt_mask>
Input event trace mask.

<non_prim_mask>
Non-primitive trace mask.

8.13 IS41 Configuration Commands

The IS41 configuration commands are:

- IS41_TRACE Command

8.13.1 IS41_TRACE Command

Synopsis

This command sets the IS41 trace masks. Refer to *IS41 Programmer's Manual* for full details.

Syntax

IS41_TRACE <op_evt_mask> <ip_evt_mask> <non_prim_mask>

Example

IS41_TRACE 0xf 0xf 0xff

Parameters

<op_evt_mask>
Output event trace mask.

<ip_evt_mask>
Input event trace mask.

<non_prim_mask>
Non-primitive trace mask.

8.14 SIGTRAN Protocol Configuration Overview

This section gves an overview of the SIGTRAN configuration model and defines the configuration parameters.

8.14.1 SIGTRAN M3UA ASP, Host to SGP Configuration Model

M3UA SIGTRAN hosts use the IETF SIGTRAN M3UA protocol to carry SS7 MTP3 traffic to and from M3UA Signaling Gateways (SG). The host connects to one or more Signaling Gateways using SIGTRAN SCTP Associations. The host can then send MTP3 messages generated by an MTP3 User Part to a remote SS7 point code via the SG. M3UA will select which SG to send these messages to depending on the Destination Point Code (DPC) of the message.

Each SG the host connects to will be configured using the SIGTRAN Link Initiate 'SNSLI' command. This identifies details of the IP link connecting the host and SG, with the host end always marked as 'client'. This command also identifies the type of SS7 traffic (e.g. ITU14). There may be up to four SNLINKs to the same SG, allowing for loadsharing of traffic.

Each DPC to which the host sends traffic must be configured using a SIGTRAN Route Initiate 'SNRTI' command. This defines a route to the DPC which must then be added to the specific Signaling Gateways that provide that route using the SIGTRAN Route List Initiate 'SNRLI' command.

M3UA uses the term Application Server (AS) to identify a host application processing SS7 traffic. The SIGTRAN Application Server Initiate 'SNAPI' command is used to configure a Local AS. Up to four Local AS are supported per host, each may be individually managed and controlled. Each LAS can be active across one or more associations or multiple LAS may share the same association, in which case Routing Context parameters must be used.

The SG maintains a 1:1 mapping of Routing Key to LAS and optionally a 1:1 mapping of Routing Key to Routing Context (RC). The Routing Key may be predefined on the SG or may be configured by the host when the link is established using Routing Key Management procedures (RKM). The SIGTRAN Routing Key Initiate 'SNRKI' command is used to define the Routing Key and the SIGTRAN Local AS Bind Initiate 'SNLBI' command is used to bind the LAS to the SG, optionally with the Routing Key id, or RC.

When establishing communication with the SG, if RKM procedures are enabled the Routing Key will be registered with the SG which will return a RC to use in traffic for that LAS-SG relationship. When going active, the local 'Traffic Mode' is used to tell the SG how to distribute traffic across multiple hosts or multiple LAS.

The system can be made to automatically audit Signaling Gateways to ensure that route status is kept synchronized with the Signaling Gateways. Auditing happens on recovery of the Signaling Gateway connection, and every 30 seconds thereafter until all destinations are available and uncongested.

8.14.2 SIGTRAN M3UA IPSP, Peer to Peer Configuration Model

M3UA can be used to allow remote end points to exchange MTP3 messages directly using SCTP Associations without using a Signaling Gateway. One or more Remote Application Servers (RAS) can be configured, each of which will have a different Point Code. M3UA will select the RAS to send the MTP3 message to depending on the Destination Point Code (DPC) of the message.

One or more SIGTRAN Links can be created with the 'SNSLI' command identifying the IP details and SS7 traffic type (e.g. ITU14). One end of the association should be marked as 'client' and the other as 'server'.

Each Remote Application Server (RAS) is defined using the SIGTRAN Remote AS Initiate 'SNRAI' command and will have a DPC and an optional Routing Context (RC).

The SIGTRAN links are attached to the RAS with the SIGTRAN AS List initiate 'SNALI' command. Up to four SNLINKs may be attached to the same RAS allowing traffic to be load shared across the associaitons. Alternatively one SNLINK may be attached to multiple RAS allowing them to share the association.

The SIGTRAN Application Server Initiate 'SNAPI' command is used to configure a Local AS. Up to four Local AS are supported per host, each may be individually managed and controlled. Each LAS can be active across one or more associations or multiple LAS may share the same association.

SIGTRAN Local AS Bind Initiate 'SNLBI' command is used to bind the LAS to the RAS with an optional RC to use for that LAS-RAS relationship. Where multiple LAS or RAS use the same association, the RC parameter must be used to differentiate traffic.

8.14.3 SIGTRAN M3UA User Parts

The default LAS1 (NC0) is configured with user parts and service indicators according to the protocols configured in the config.txt file or explicitly by using the MTP_USER_PART NC0 command. For each additional LAS configured the MTP_USER_PART NCx command must be used to connect the LAS to its user part which may be configured in a separate config.txt or externally with DSI messages.

8.14.4 M2PA Configuration Model

M2PA SIGTRAN hosts use MTP3 as normal. Instead of using SS7 Signaling Boards they use IP and SIGTRAN SCTP to carry SS7 traffic. Each M2PA link is configured using a 'SNSLI' command and an MTP_LINK command.

A single M2PA link may carry the same load as a number of MTP2 links. This means a typical Link Set should only require a single M2PA link.

8.14.5 SIGTRAN SUA IPSP, Peer to Peer Configuration Model

SUA SIGTRAN hosts use the IETF SIGTRAN SUA protocol to carry SS7 SCTP traffic to a Remote Application Server. Signaling links to a RAS are instantiated using the SNSLI command. This gives details of the IP link connecting the host to the RAS. This command also identifies the type of SS7 traffic to be used (e.g. ITU14). This command is used once for each Signaling link that is to be supported.

Each LAS is identified with a SNAPI command. A maximum of four LASs are supported. Each RAS must be identified with a SNRAI command. A maximum of 32 RASs are supported.

The SNLBI command is used to bind a LAS to a RAS or RSG. The SNALI command is used to bind an RAS to a specific link.

The SNRTI command is used to define a route. Each route is bound to a specific RAS or RSG by an SNRLI command. A maximum of 64 routes and route bind commands are supported.

The DPC must either be defined in the SNRAI command which defines the RAS or in any route which is subsequently bound to the RAS.

8.14.6 SIGTRAN SUA ASP, Host to SGP Configuration Model

SUA SIGTRAN hosts use the IETF SIGTRAN SUA protocol to carry SS7 SCTP traffic to SUA Remote Signaling Gateways. Links to the RSG are declared using the SNSLI command. This gives details of the IP link connecting the host to the RSG and declares the presence of an RSG by including an SG parameter.

Each LAS is identified with a SNAPI command. A maximum of four LASs are supported. RSGs do not have to be explicitly declared – they are set up implicitly by the SNSLI command.

The SNLBI command is used to bind a LAS to an RSG. The SNALI command is not required as the RSG is bound to a specific link by the SNSLI command.

As in IPSP configurations, the SNRTI command is used to define a route and routes are bound to a specific RSG by an SNRLI command. A maximum of 64 routes and route bind commands are supported.

For SG connections, the DPC must be included in any routes which are bound to the RSG.

8.14.7 SIGTRAN Parameters

The Configuration of SCTP, SCTPD, SCTPN, M3UA and SUA uses Man Machine Interface (MMI) format commands. MMI commands start with a 5 character command name; if parameters are included, then the command name is followed by a colon and then the parameters. Parameters are of the format 'parameter name'='parameter value' and are separated by a comma. The command line is terminated with a semi-colon.

The following parameters are supported:

Name	Description	Range	Default
AS	Application Server	1:4	
AUTOACT	Automatic activation of SIGTRAN associations	Y, N	Y
CIC_RANGE	Range of ciscuits for use with ISUP/TUP. This is a compound parameter comprising the CIC of the first circuit ('base') and the number of consecutive circuits included ('range').	<base>-<range>	
DAUD	Destination Audit	Y, N	N
DPC	Destination Point Code	0:16777215	
DUAL	Dual resilient configuration host identifier	A,B	
HPORT	Host SCTP Port (M3UA) (SUA)	1:65535	2905 14001

Name	Description	Range	Default
HIPADDR1	First per-link (local) host IP address.	IPv4 addresses using dot notation: w.x.y.z Or IPv6 addresses using colon notation A:B:C::E:F.	
HIPADDR2	Second per-link (local) host IP address.		
HIPADDR3	Third per-link (local) host IP address.		
HIPADDR4	Fourth per-link (local) host IP address.		
IPADDR	IP address of Host or SG or RAS.	IPv4 addresses using dot notation: w.x.y.z Or IPv6 addresses using colon notation A:B:C::E:F.	
IPADDR2	Second IP address of Host or SG or RAS.		
IPADDR3	Third IP address of Host or SG or RAS.		
IPADDR4	Fourth IP address of Host or SG or RAS.		
M2PA	Logical reference for an M2PA Link	1:8	
M2PA_VER	Version of M2PA Protocol to support	RFC, 9	RFC
M3UAHBT	M3UA Heartbeats enable	Y,N	N
MAXSIF	Max Signaling Information Field accepted in API_MSG_TX_REQ	5:4199	272
MODULE	Name of module to configure	DMR, DTC, DTS, INAP, IS41, ISUP, M2PA, M3UA , MAP, MTP3, RMM, SCCP, SCTP, SCTPD, SUA, TCAP	
MOD_ID	Module ID	0:255	
NA	Network Appearance	0:4294967295	
NC	Network Context (maps to LAS)	0:3	0
NASP	Minimum number of ASPs required to fully resource the AS.	0:64	0
OPC	Originating Point Code	0:16777215	
OPTIONS	Run-time options used in SNRLI, SNRTI CNNCI and CNOPS commands	0: 0xffffffff	
PER	Personality Configuration	0:255	0
PPORT	Peer SCTP Port (M3UA) (SUA)	1:65535	2905 14001
RAS	Remote Application Server	1:64	
RC	SIGTRAN Routing Context	0:4294967295	
RTXB	Max SCTP heartbeat retransmissions (max_retx_heartbeat)	0:20 0 = use SCTP default	2
RTXD	Max SCTP data retransmissions (max_retx_data)	0:20 0 = use SCTP default	2
RTXI	Max SCTP init retransmissions (max_retx_init)	0:20 0 = use SCTP default	5
RTXP	Number of consecutive init retransmissions on a path	0:20 0 = use SCTP default	1
SG	Logical reference for a Signaling Gateway	1:64	

Name	Description	Range	Default
SI	Service Indicator	0:15	
SNAL	Logical identifier of a SIGTRAN Remote Application Server list	1:256	
SNEND	Determines Client or Sever for SCTP	C,S	C
SNLB	Logical ID of SIGTRAN AS Bind	1:256	
SNLINK	Logical ID of a SIGTRAN Link	1:384	
SNMP	SNMP enabled for object	Y,N	N
SNRK	Logical ID of a M3UA Routing Key	1:64	
SNRL	Logical ID of a SIGTRAN Route List	1:512	
SNRT	Logical ID of SIGTRAN Route	1:4096	
SNTYPE	Type of SNLINK	M2PA, M3UA, SUA, DMR	M3UA
SS7MD	SS7 variant.	ITU14, ITU16, ITU24, ANSI	ITU14
SSN	Sub-System Number	0:255	0
SSR_ID	Sub-system resource to be used by this LAS	0:511	0
TID_START	Start bit for TCAP instance identifier	0:31	
TID_END	End bit for TCAP instance identifier	0:31	
TID_VALUE	Value of TCAP instance identifier	0:65535	
TMSEC	Time value in milliseconds (ms)	0:15000	
TRMD	Traffic Mode for Host system Load Share – LS Broadcast – BC Override - OR	LS, BC, OR	LS
TSEC	Time value in seconds	0:65	
TTYPE	Protocol Timer type	SCTP,M2PA,M3UA	

8.14.8 IP address scope

The IPv4 address space is divided into four different scopes:

- Global – these are the globally unique addresses used on the public Internet.
- Site-local – there are three site-local networks which are defined by RFC 1918. These are 10.0.0.0/8, 172.16.0.0/12 and 192.168.0.0/16. Site-local addresses are only unique within the local network and are therefore meaningless to systems outside of the local network.
- Link-local – there is one link-local network, 169.254.0.0/16, which is defined by RFC 3927. Link-local addresses are only unique within the link and are therefore meaningless to external systems.
- Loopback – the loopback network, 127.0.0.0/8, is defined by RFC 1700 and is only valid within the local host.

The IPv6 address space is divided into three different scopes:

- Global – these are the globally unique addresses used on the public Internet.

- Link-local – for uniquely identifying interfaces within (i.e., attached to) a single link only. Link-local addresses are only unique within the link and are therefore meaningless to external systems.
- Loopback – The IPv6 unicast loopback address, ::1, is treated as having link-local scope within an imaginary link to which a virtual "loopback interface" is attached.

When communicating with a host, addresses from a more restrictive scope must never be transmitted since it is likely that such an address would be meaningless (or even harmful) to the remote system. Since this behavior is enforced on some SCTP implementations, it is recommended that addresses of different scopes not be mixed within a single multi-homed association.

8.15 SIGTRAN Configuration Commands

The SIGTRAN configuration commands are:

- SNAPI Command - SIGTRAN Local AS
- SNSLI Command - SIGTRAN Signaling Link Initiate
- SNRTI Command - SIGTRAN Route
- SNRLI Command - SIGTRAN Route List
- SNRKI Command - SIGTRAN Routing Key Initiate
- SNALI Command - SIGTRAN AS List Initiate
- SNLBI Command - SIGTRAN Local AS Bind Initiate
- CNSYS Command - Configuration System Set
- CNOPS Command - Configuration Module Options Set
- CNNCI Command - Configuration Network Context Initiate
- CNTOS Command - Configuration Timeout Set

8.15.1 SNAPI Command - SIGTRAN Local AS Initiate

Synopsis

This command initiates a Local Application Server (LAS). A LAS is a logical entity representing an SS7 end point that can process circuit-related and or non circuit-related signaling. Communication with a SG or Remote Application Server may use the Routing Context to identify the LAS.

SS7MD defaults to ITU14 and the local traffic mode (how peer should route traffic to this LAS) defaults to loadshare. Optional OPC is for information only.

TID parameters are for SUA only and if any TID parameter is present they must all be present.

For M3UA the Routing Context parameter is optional and then only used when connecting to a remote SG. When connecting to a Remote AS the Routing Context parameter can be configured by SNRAI. In most cases it is recommended that the Routing Context parameter be configured by SNLBI.

Syntax

SNAPI:LAS=,[RC=,][SS7MD=,][TRMD=,][SNTYPE=,][OPC=,]
[TID_START=,][TID_END=,][TID_VALUE=,][SSR_ID=];

Example

SNAPI:LAS=1,OPC=123;
SNAPI:LAS=2,SS7MD=ITU16,TRMD=OR;

Prerequisites

The RC, if specified, must not already be associated with another local AS

8.15.2 SNSLI Command - SIGTRAN Signaling Link Initiate

Synopsis

This command initiates a SIGTRAN link. A SIGTRAN link provides an SCTP association to a Signaling Gateway or Remote Application Server Process or between two M2PA nodes, or Diameter nodes.

If two IP addresses are specified, then the first IP address will be used until it proves unreliable in which case the second will be used.

SS7MD defaults to ITU14 if not specified.
SNTYPE defaults to M3UA if not specified.

For M3UA, M3UAHBT (Heartbeat) and SNMP indications may be enabled.

SG must be specified for a link to a Signaling Gateway, otherwise this is an IPSP link.

If no M2PA Version is specified and the link type is M2PA, then it will default to the RFC version.

Syntax

SNSLI:SNLINK=,IPADDR=,[SG=,][SS7MD=,][IPADDR2=,][IPADDR3=,]
[IPADDR4=,][HIPADDR1=,][HIPADDR2=,][HIPADDR3=,][HIPADDR4=,]
[HPORT=,][PPORT=,][SNEND=,][SNTYPE=,][M2PA=,][M2PA_VER=,]
[NA=,][M3UAHBT=,][SNMP=,][RTXI=,][RTXP=,][RTXD=,][RTXB=];

Example

SNSLI:SNLINK=1,SNEND=C,IPADDR=192.168.0.1,SG=2, IPADDR2=192.168.1.1;

SNSLI:SNLINK=2,IPADDR=192.168.0.10,HPORT=2906,PPORT=2906,
SNEND=S,SNTYPE=M3UA;

SNSLI:SNLINK=3,IPADDR=192.168.0.20,SNEND=C,SNTYPE=M2PA,M2PA=3,
PPORT=3567;

SNSLI:SNLINK=1,IPADDR=2001:DB8::1234:5678,HIPADDR1=2001:DB8::1234:ABCD
,SNEND=S,SNTYPE=DMR,PPORT=3868,HPORT=3868;

Prerequisites

SS7MD must be the same throughout the system

For security reasons, the user must explicitly specify the IP addresses the link is connected to. The Peer is not allowed to respond or request IP addresses not configured by this command.

An IP address of 0.0.0.0 cannot be specified

8.15.3 SNRTI Command - SIGTRAN Route Initiate

Synopsis

This command is used to configure a SIGTRAN route to a remote SS7 destination.

For M3UA: if the LAS parameter is not specified the route is attached to LAS1 by default. The DPC is mandatory and SNMP indications are available.

For SUA: if the LAS parameter is not specified then the route is attached to all LASs

Syntax

SNRTI:SNRT=,[DPC=,][LAS=,][SNTYPE=,][SSN=,][OPTIONS=,][SNMP=];

Example

SNRTI:SNRT=1,DPC=130;
SNRTI:SNRT=2,DPC=200,LAS=2,OPTIONS=0x0004;

Prerequisites

Another route cannot exist with the same DPC.

Parameters

<OPTIONS>

The OPTIONS parameter is a 16 bit field to configure run-time options with values defined below. If omitted, it defaults to 0x0002 for backwards compatibility

Bit	Mnemonic	Meaning
0	M3UOP_ROUTE_ASSUME_AVAIL	Assume route always available
1	M3UOP_ROUTE_LOADSHARE	Loadshare across all servers in the route
2	M3UOP_ROUTE_DEFAULT	Default Route
3	M3UOP_ROUTE_SNMP	Enable SNMP indications for this route (will also be set by specifying SNMP='Y')

8.15.4 SNRLI Command - SIGTRAN Route List Initiate

Synopsis

This command attaches Signaling Gateways to a SIGTRAN Route. A SIGTRAN route will use these adjacent Signaling Gateways to reach an eventual destination Point Code.

For SUA a RAS or a SG must be specified.

For M3UA SG is mandatory and OPTIONS may be specified. Where a route (SNRT) is attached to more than 1 SG, and route loadshare is enabled, this applies to the first 2 active SGs in the SNRL list.

Syntax

SNRLI:SNRL=,SNRT=,[SG=,][RAS=,][OPTIONS=];

Example

SNRLI:SNRL=1,SNRT=1,SG=1;

Prerequisites

The route has already been initiated.

A SG must have at least one SNLINK associated with it. For SUA if SG is specified the route must contain DPC. If RAS is specified the DPC must be in each route or the RAS.

Parameters

<OPTIONS>

The OPTIONS parameter is a 16 bit field to configure run-time options with values defined below. If omitted, it defaults to 0.

Bit	Mnemonic	Meaning
0	M3UOP_ROUTE_SERV_ASSUME_AVAIL	Presume the route is available via the server once the server is available, i.e. without waiting for reception of DAVA

8.15.5 SNRKI Command - SIGTRAN Routing Key Initiate

Synopsis

This command initiates a Routing Key for use with a Signaling Gateway (ASP Mode). Once defined the SNRK id may be used in an SNLBI command in place of the RC parameter to add the Routing Key to a Signaling Gateway. At runtime the Routing Key will be registered with the Signaling Gateway which will return the RC to use.

The SNRK parameter is the logical identifier of the Routing Key (in the range 1 to 64).

CIC_RANGE is a compound parameter in the form <base-range> where base is the base CIC and range is the number of contiguous CICs in the range.
e.g. CIC_RANGE 10-15 results in cics 10 to 24 inclusive. When CIC_RANGE is specified the OPC parameter is mandatory.

When specifying OPC and DPC they are specified from the Signaling Gateway's point of view, i.e. OPC is the remote point code that is originating traffic and DPC represents the Local AS which intends to receive the messages.

Syntax

SNRKI:SNRK=,DPC=,[SI=,][OPC=,][OPC=,CIC_RANGE=];

Example

SNRKI:SNRK=1,DPC=123;
SNRKI:SNRK=2,DPC=123,OPC=567;
SNRKI:SNRK=3,DPC=123,OPC=567,SI=5,CIC_RANGE=64-32;

Prerequisites

Can only be used when connecting to a Signaling Gateway (not IPSP mode)

8.15.6 SNRAI Command - SIGTRAN Remote AS Configuration

Synopsis

This command initiates a Remote Application Server entity. A Remote Server represents a Remote SS7 signaling point. A Remote AS may run on a number of remote hosts. The optional NASP parameter defines the target number of RAS's required in load sharing mode.

SS7MD defaults to ITU14.

TRMD is the Peer Traffic Mode (ie how traffic is routed towards the peer), it defaults to LS (Loadshare).

For SUA if the DPC is omitted, it must be specified in all SNRTs bound to this RAS.

For M3UA the (optional) Routing Context parameter for use with the RAS is configured by this command, not by the SNAPI command.

Syntax

SNRAI:RAS=,[RC=,][DPC=,][NASP=,][TRMD=,][SNMP=,][LAS=,];

Example

SNRAI:RAS=1,RC=1,DPC=555;

Prerequisites

RC must not be associated with any other RAS.

Normally only one RAS or SNRT can be configured with a particular DPC, however for M3UA multiple RAS may be configured with the same DPC providing a different LAS is specified.

A remote application server may only be bound to a single local application server.

8.15.7 SNALI Command - SIGTRAN AS List Initiate

Synopsis

This command is used to attach a SIGTRAN link to a Remote Application Server. It identifies the remote ASPs that the AS is hosted on.

Syntax

SNALI: SNAL=,RAS=,SNLINK=;

Example

SNALI:SNAL=1,RAS=1,SNLINK=1;

Prerequisites

The Remote Application Server has already been initiated.

The specified SIGTRAN link has already been initiated.

The SIGTRAN link must be of type M3UA or SUA and connect to a Remote ASP (IPSP).

For M3UA an SNLINK may be attached to multple RAS.

The application list id SNAL must not have been used.

For SUA an automatic SNALI will be created for any SNLINK to a SG. The SNAL id will be the same as the SNLINK id and therefore may not be used for other SNALI commands.

A RAS must have at least one SNLINK associated with it.

8.15.8 SNLBI Command - SIGTRAN Local AS Bind Initiate

Synopsis

This command initiates a relationship between a Local Application Server and Remote Application Server or Signaling Gateway. The Local Application Server will use the SIGTRAN Links associated with the Remote Application Server or Gateway.

For M3UA IPSP use this command is not required.

For ASP to SG configurations this command will use the RC parameter from the LAS (see SNAPI) and the SG may only be bound to one LAS.

Syntax

SNLBI: SNLB=,LAS=,[RAS=,][SG=,][RC=,][SNRK=];

Example

SNLBI:SNLB=1,LAS=1,SG=1;
SNLBI:SNLB=2,LAS=2,RAS=1;
SNLBI:SNLB=3,LAS=3,RAS=2,RC=2;
SNLBI:SNLB=4,LAS=4,SG=2,SNRK=1;

Prerequisites

The LAS and RAS or SG have been initiated.

The RAS or SG is associated with at least one SNLINK.

The RAS or SG is not attached to another LAS.

Routing Keys (SNRK) may only be specified when binding to a SG

8.15.9 CNSYS Command - Configuration System Set

Synopsis

This command allows system wide settings to be configured.

Checksums for SCTP associations default to CRC32. If ADLER is required then PER should be set to 1.

If DAUD is set to 'Y' then each SG will be audited concerning route status.

For Dual Resilient systems using RMM the DUAL parameter must be applied with value 'A' or 'B'. This parameter may be used in non-SIGTRAN configurations.

The AUTOACT parameter can be set to 'N' to disable automatic activation of SIGTRAN associations. For M3UA when set to 'Y' (default) the following 3 steps occur:

1. Association is activated at SCTP level
2. ASP is brought UP (when possible)
3. ASP is made ACTIVE (when possible)

If set to 'N' it is the users responsibility to control the steps as required with GCT messages. This allows users to activate the association only and have a peer control the ASPUP and ASPAC stages.

IPADDR is mandatory for SIGTRAN, optional for TDM configurations.

SNMP setting enables SNMP for all objects in the system.

Syntax

CNSYS: [IPADDR=,][IPADDR2=,][IPADDR3=,][IPADDR4=,][PER=,] [DAUD=,][DUAL=,][AUTOACT=,][SNMP=];

Example

CNSYS:IPADDR=192.168.1.20;
CNSYS:IPADDR=192.168.1.20,DAUD=Y;
CNSYS:IPADDR=2001:DB8::1234:5678,DUAL=A,AUTOACT=N;

8.15.10 CNOPS Command - Configuration Module Options Set

Synopsis

This command allows per-module settings to be varied from the default settings. Specifically it allows the default module_id to be set to a different value and, for certain modules, allows additional run-time configuration options to be set.

Syntax

CNOPS: MODULE=,[OPTIONS=,][MOD_ID=,][NC=];

Example

CNOPS: MODULE=M2PA,OPTIONS=0x0123;
CNOPS: MODULE=MTP3,MOD_ID=0x3d;
CNOPS: MODULE=M3UA,MOD_ID=0xfd,NC=1;

Parameters

<MODULE>

A token representing the module for which the configuration applies. Possible values are: DMR, DTC, DTS, INAP, IS41, ISUP, M2PA, M3UA, MAP, MTP3, RMM, SCCP, SCTP, SCTPD, SUA and TCAP.

<MOD_ID>

The value for the module_id in the range 0x01 to 0xfe. Care should be taken when selecting module_id value to ensure that the value is not already in use. Typically it is not necessary to change the default module_id.

<NC>

Network Context. Defaults to 0 if not specified.

<OPTIONS>

This is a 32 bit value used to set the per-module run-time configuration options. This parameter is only valid when used with the following settings for <MODULE>: M2PA, M3UA, RMM, SCTP and SCTPD. Other modules have the ability to set the run-time options in other config.txt commands.

M2PA Options

Bit	Meaning
0	Set to 1 to use Multiple Congestion levels
1	Set to 1 to use 7-bit sequence numbers instead of default 24-bit sequence numbers. Use if MTP3 does not support Extended Changeover Procedures.
2	Set to 1 to use the (older) Draft Version 9 of the M2PA Specification. Default operation supports the RFC specification.

M3UA Options

Bit	Meaning
0	Enable IPSP functionality
1	Enable Signaling Gateway functionality
2	Set to 1 to select the lowest bit of the SLS value to determine which Signaling Gateway to route traffic to. If not set, the highest bit of the SLS value is used.
3	By default, data traffic is load-shared across the SCTP streams based on the SLS value. When set, this option forces the M3UA module to use only stream 1 for transmitting data.

RMM Options

Bit 0	Bit 1	Meaning
0	0	14 bit point codes
1	0	16 bit point codes
0	1	24 bit point codes

SCTP Options

Bit	Meaning	
	SCTP/SCTPD	SCTPN
0	Controls the SCTP checksum algorithm. When set to 1 the CRC32 checksum is used otherwise the Adler32 (FRC2960) checksum is used.	When using the native SCTP module (SCTPN) Adler32 checksum is not supported so this bit should always be set to 1.
1	Module will ignore rather than abort incoming connection attempts for none present SCTP ports.	Reserved for future use and should be set to 0.
4	Forces retransmits of the data on the same path until it is considered inactive	Reserved for future use and should be set to 0.
5	If set the SCTP module will use the preferred path if available. The preferred path uses the first host address set for the association.	If set the SCTPN module will use the preferred path if available. The preferred path uses the first host address set for the association.

8.15.11 CNNCI Command - Configuration Network Context Initiate

Synopsis

This command configures basic network variant and configuration options for a network context.

Syntax

CNNCI:SS7MD=,[OPTIONS=,][NC=,][MAXSIF=];

Example

CNNCI:SS7MD=ANSI,OPTIONS=0x0003;
CNNCI:SS7MD=NC=1,ITU14,MAXSIF=272;

Parameters

<SS7MD>

The SS7 network variant. Takes one of the following values: ANSI, ITU14, ITU16 or ITU24. If SS7MD is specified it must be consistent with the setting of this value elsewhere within the same Network Context, e.g. the SNAPI configuration command.

<NC>

Network Context. Defaults to 0 if not specified.

<MAXSIF>

Maximum permitted number of octets in the Signaling Information Field for transmission to the network. If omitted defaults to 272.

<OPTIONS>

The OPTIONS parameter is a 16 bit field to configure run-time options with values defined below. If omitted, it defaults to 0.

Bit	Meaning
0	Set to 1 to activate M3UA SLS Rotation. Used in conjunction with bit 1 (see below)
1	When M3UA SLS rotation is enabled (see bit 0) this bit controls the number of SLS bits that are rotated as follows: Set to zero for 4 or 5 bit SLS rotation based on protocol variant. Set to 1 for 8 bit SLS rotation.

8.15.12 CNTOS Command - Configuration Timeout Set

Synopsis

This command allows the user to set the values of timers to be used in the SCTP, M2PA and M3UA protocols.

Syntax

CNTOS:TTYPE=,TO=,{TSEC=|TMSEC=};

Examples

CNTOS:TTYPE=SCTP,TO=HBT,TMSEC=500;
CNTOS:TTYPE=M2PA,TO=T4N,TSEC=10;

Parameters

<TTYPE>
The protocol timer type, SCTP, M2PA or M3UA.

<TSEC>, <TMSEC>
The value of the timeout in seconds or milliseconds.

<TO>
The token designating a particular timer taken from the following table:

TO	Default	Granularity	SCTP Timer
Rmin	200ms	1ms	Minimum RTO
Rmax	1400ms	1ms	Maximum RTO
Rinit	1000ms	1ms	Initial RTO
Ck	30000ms	1ms	Cookie lifetime
Hbt	1000ms	1ms	Time between heartbeats
T1i	3000ms	1ms	Starting timeout of an INIT chunk
T2i	3000ms	1ms	Starting timeout of a SHUTDOWN chunk

SACKD	10ms	1ms	SACK delay timer (Linux only) For Solaris the SACK delay time can be adjusted if required using operating system utilities as follows:
			`ndd -set /dev/sctp`
			`sctp_deferred_ack_interval 10`

TO	Default	Granularity	M2PA Timer
T1	40s	1s	'Alignment Ready' timer value
T2	10s	1s	'Not Aligned' timer value
T3	2s	1s	'Aligned' timer value
T4n	7s	1s	'Normal Proving' timer value
T4e	500ms	100ms	'Emergency Proving' timer value
T6	3s	1s	'Remote Congestion' timer value
T7	1s	100ms	'Excessive Delay Of Acknowledgement' timer value
TO	**Default**	**Granularity**	**M3UA Timer**
Tack	2000ms	1ms	Peer response timeout
Tr	1000ms	1ms	Recovery timer for inactive ASPs
Tdaud	30s	1s	DAUD generation timer
Tbeat	30s	1s	M3UA heartbeat timer

8.16 Diameter Parameters

Configuration of the Diameter protocol uses Man Machine Interface (MMI) format commands. MMI commands start with a 5 character command name; if parameters are included, then the command name is followed by a colon and then the parameters. Parameters are of the format 'parameter name'='parameter value' and are separated by a comma. The command line is terminated with a semi-colon.

The following parameters are supported:

Name	Description	Range	Default
APPID	The Diameter Application ID AVP value for use in Diameter capability negotiation.	S6a, S13, Rf, NAS, CC, ACCT	
BASEICD	First session ID for incoming sessions	0: 0xffffffff	
BASEOGD	First session ID for outgoing sessions	0: 0xffffffff	
DMAP	Logical ID of a Diameter Application	0:15	
DMNC	Logical ID of a Diameter Network Context	0:3	
DMPR	Logical ID of a Diameter Peer	0:255	
DMRL	Logical ID of a Diameter Route List	0:2047	
DMRT	Logical ID of a Diameter Route	0:1023	
HOST	Diameter Host-name AVP value	FQDN	
MOD_INST	Module Instance	0:255	
NODENAME	Diameter Node-name AVP value.A logical node label string.		
POLICYID	Identifier providing a routing policy for the route.	0:0xffff	
REALM	Diameter Realm-name AVP value.	FQDN	
VENDORID	The Diameter Vendor ID AVP value for use in Diameter capability negotiation.	0: 0xffffffff	

8.17 Diameter Configuration Commands

The Diameter configuration commands are:

- DMNCI Command - Diameter Network Context Initiate
- DMPRI Command - Diameter Peer Initiate
- DMRTI Command - Diameter Route Initiate
- DMRLI Command - Diameter Route List Initiate
- DMAPI Command - Diameter Application Initiate
- DMSYI Command - Diameter System Initiate

8.17.1 DMNCI Command - Diameter Network Context Initiate

Synopsis

Command to initiate a Diameter Network Context. This command allows the node name, origin host and realm as well as additional options to be set for the Diameter Node.

If not specified, the DMNC defaults to zero.

Syntax

DMNCI:DMNC=,[OPTIONS=,]HOST=,REALM=,NODENAME=;

Example

```
DMNCI:DMNC=0,OPTIONS=0x00,HOST=dmr01.lab.dialogic.com,
      REALM=dmradmin01.dialogic.com,
      NODENAME=ExampleMME
```

Prerequisites

The DMNC value must be unique within the system.

8.17.2 DMPRI Command - Diameter Peer Initiate

Synopsis

Command to configure parameters of the Diameter Peer node to be specified. The SNLINK parameter specifies the SCTP association which will be used to communicate with the Peer. The DMNC parameter indicates which Diameter Network Context should handle traffic for this Peer.

If not specified, the DMNC defaults to zero.

Syntax

DMPRI:DMPR=[,DMNC=][,OPTIONS=],SNLINK=,HOST=,REALM=[,LABEL=];

Example

DMPRI:DMPR=10,DMNC=0,OPTIONS=0x00000000,SNLINK=1,HOST=dmr02.lab.dialogic.com,REALM=dmradmin01.dialogic.com,LABEL=Paris;

Prerequisites

An SNLINK may only be specified by one Peer. DMPR value must be unique.

8.17.3 DMRTI Command - Diameter Route Initiate

Synopsis

Command to initiate a Diameter Route. The Diameter Route defines a final Peer or remote node reachable via a Relay agent.

If not specified, the DMNC, OPTIONS, POLICYID and APPID default to zero. The default route option allows a single route to be intentified for use when no specific host or realm is matched. Realm-based routing allows routing to a set of hosts that match a specific realm. Host-based routing will only route to a specific host.

Syntax

Default Routing:

DMRTI:DMRT=,OPTIONS=[,DMNC=][,LABEL=];

Host Routing:

DMRTI:DMRT=,HOST=[,DMNC=][,OPTIONS=][,POLICYID=][,LABEL=];

Realm Routing:

DMRTI:DMRT=,REALM=[,DMNC=][,OPTIONS=][,APPID=][,POLICYID=][,LABEL=];

Examples

DMRTI:DMRT=1,DMNC=0,OPTIONS=0x00000001,LABEL=DefaultRoute;

DMRTI:DMRT=5, ,DMNC=0,HOST=dmr02.lab.dialogic.com,LABEL=HostRoute;

DMRTI:DMRT=35,DMNC=0,APPID=S6A,REALM=dialogic.com,POLICYID=10,LABEL=RealmRoute;

Prerequisites

DMRT value must be unique. The Route configuration must contain one instance of HOST, REALM or Default Route option.

8.17.4 DMRLI Command - Diameter Route List Initiate

Synopsis

Command to initiate a Diameter Route List entry. The Diameter Route List identifies the Peer which can be used by a Route.

If not specified, the DMNC defaults to zero.

Syntax

DMRLI:DMRL=,[DMNC=,]DMPR=,DMRT=;

Example

DMRLI:DMRL=5,DMNC=0,DMPR=10,DMRT=5;

Prerequisites

DMRL value must be unique. The DMRT parameter must identify the ID of a configured Route.

8.17.5 DMAPI Command - Diameter Application Initiate

Synopsis

Command to specify the applications which are advertised or accepted during capabilities exchange. This command should be specified for each Diameter Application required, for the specified DMNC.

If not specified, the DMNC defaults to zero.

Syntax

```
DMAPI:DMAP=[,OPTIONS=],MOD_ID=[,MOD_INST=][,DMNC=],
      APPID=,VENDORID=[,LABEL=];
```

Example

```
DMAPI:DMAP=1,OPTIONS=0x00000000,MOD_ID=0x2d,MOD_INST=0,
      DMNC=0,APPID=S6a,VENDORID=test,LABEL=Billing;
```

Prerequisites

DMAP value must be unique.

8.17.6 DMSYI Command - Diameter System Initiate

Synopsis

This command allows diameter module settings to be varied from the default settings. Specifically allows additional run-time configuration options to be set.

Syntax

```
DMSYI:[BASEICD=,][BASEOGD=,][OPTIONS=,];
```

Example

```
DMSYI:BASEICD=0x00,BASEOGD=0x8000;
```

Parameters

<BASEICD>
<BASEOGD>

Set the value for the base incoming and outgoing session IDs for the Diameter module. If these values are not specified the default values of 0x0000 and 0x8000 will be used. The module will then allow session IDs of <BASEICD> to <BASEOGD> -1 for incoming sessions. And <BASEOGD> to 0xfffe for outgoing sessions. Note: the value of 0xffff is reserved.

9 Example Configuration Files

9.1 Example system.txt System Configuration file

```
*******************************************************************************
*
* Example System Configuration File (example_system.txt) for use with
* the Dialogic(R) DSI Development Package.
*
* Edit this file to reflect your configuration.
*
*******************************************************************************
*
* Essential modules running on host:
*
LOCAL           0x20            * ssds/ssdh/ssdm - Board interface task
LOCAL           0x00            * tim - Timer task
*
* Optional modules running on the host:
*
LOCAL           0xcf            * s7_mgt - Management/config task
LOCAL           0xef            * s7_log - Display and logging utility
LOCAL           0x2d            * upe - Example user part task
*
* Modules that optionally run on the host:
*
LOCAL         0x22              * MTP3 module (and SS7LD 'mtp' and 'isup' run-mode)
* LOCAL         0x23            * ISUP module (and SS7LD 'isup' run-mode)
* LOCAL         0x4a            * TUP module
* LOCAL         0x33            * SCCP module
* LOCAL         0x14            * TCAP module
* LOCAL         0x15            * MAP module
* LOCAL         0x25            * IS41 module
* LOCAL         0x35            * INAP module
*
* Essential modules running on the board (all redirected via ssd):
*
REDIRECT        0x71    0x20    * MTP2 module (except SS7HD boards)
* REDIRECT      0x81    0x20    * MTP2 module_id for SP 0 (SS7HD boards only)
* REDIRECT      0x91    0x20    * MTP2 module_id for SP 1 (SS7HD boards only)
* REDIRECT      0xe1    0x20    * MTP2 module_id for SP 2 (SS7HD boards only)
* REDIRECT      0xf1    0x20    * MTP2 module_id for SP 3 (SS7HD boards only)
REDIRECT        0x10    0x20    * CT bus/Clocking control module
REDIRECT        0x8e    0x20    * On-board management module
*
* Modules that optionally run on the board (all redirected via ssd):
*
* REDIRECT        0x22    0x20    * MTP3 module (except for SS7LD 'mtp' and 'isup' run
modes)
* REDIRECT      0x23    0x20    * ISUP module (except for SS7LD 'mtp' and 'isup' run
modes)
* REDIRECT      0x4a    0x20    * TUP module
* REDIRECT      0x33    0x20    * SCCP module
* REDIRECT      0x14    0x20    * TCAP module
* REDIRECT      0x15    0x20    * MAP module
* REDIRECT      0x25    0x20    * IS41 module
```

```
* REDIRECT      0x35    0x20    * INAP module
*
*
* SS7MD boards only:
*
* REDIRECT      0x31    0x20    * ATM Module
* REDIRECT      0x41    0x20    * Q.SAAL Module
* REDIRECT      0x61    0x20    * Signalling Driver Module
*
* Redirection of status indications:
*
REDIRECT        0xdf    0xef    * LIU/MTP2 status messages -> s7_log
*
DEFAULT_MODULE  0xef            * Redirect messages by default to module 0xef
*
* Dimensioning the Message Passing Environment:
*
NUM_MSGS        5000            * Number of standard size
*                                 messages in the environment
*NUM_LMSGS      200             * Number of 'long' messages
*                                 (used for certain TCAP based applications)
*
* Now start-up all local tasks:
*   for SPCI start-up use ssds
*   for SS7HD boards use ssdh
*   for SS7MD boards use ssdm
*   for SS7LD boards use ssdl
*
* FORK_PROCESS   ./ssds
* FORK_PROCESS   ./ssdh
* FORK_PROCESS   ./ssdm
* FORK_PROCESS   ./ssdl
FORK_PROCESS    ./tim
FORK_PROCESS    ./tick
FORK_PROCESS    ./s7_mgt
FORK_PROCESS    ./s7_log
FORK_PROCESS    ./HSTBIN/mtp3
FORK_PROCESS    ./upe
*
*
```

9.2 Example config.txt Protocol Configuration File

```
*******************************************************************************
*
* Example Protocol Configuration File (example_config.txt) for use with
* the Dialogic(R) DSI Development Package.
*
* Boards supported are SPCI, SS7MD, SS7HD and the SS7LD range.
* Note - Not all boards are supported on all operating system.
*
* This file needs to be modified to suit individual circumstances.
* Refer to the relevant Programmer's Manuals for further details.
*
*******************************************************************************
*
*
* Configure individual boards:
* SS7_BOARD <board_id> <board_type> <flags> <code_file> <run_mode>
*
* For SPCI2S boards:
*SS7_BOARD  0  SPCI2S 0x0043  ./DC/ss7.dc3 MTP2
*
* For SPCI4 boards:
*SS7_BOARD  0  SPCI4 0x0043  ./DC/ss7.dc3 MTP2
*
* For SS7HD PCI boards:
*SS7_BOARD  0 SS7HDP 0x0043  ./DC/ss7.dc4  MTP2
*
* For SS7HD PCIe boards:
*SS7_BOARD  0 SS7HDE 0x0043  ./DC/ss7.dc4  MTP2
*
* For SS7MD boards:
*SS7_BOARD 0 SS7MD 0x0001 ./DC/ss7.dc6 LSL
*
* For SS7LD boards:
*SS7_BOARD 0 SS7LD 0x0001 ./DC/ss7.dc7 MTP2
*
TRACE_MOD_ID    0xef    * Set default trace module to 0xef.
*
* Configure individual T1/E1 interfaces:
* LIU_CONFIG <board_id> <liu_id> <liu_type> <line_code> <frame_format>
*           <crc_mode> [<build_out>]
LIU_CONFIG  0  0  5  1  1  1
*
* ATM parameters:
*
* Configure ATM module (SS7MD boards only):
* ATM_CONFIG <options> <num_streams>
*
* ATM_CONFIG 0x0000 4
*
* Define an ATM Cell Stream (SS7MD boards only):
* ATM_STREAM <id> <board_id> <cellstream_id> <liu_id> <options> <ima_frame_len>
<max_frame_len>
*            <def_vpi> <def_vci> <timeslot>
*
* ATM_STREAM  3  0  1  0  0x01  0  280  12  10  0xfffefffe
```

```
*
*
* Configure MTP3 module:
* MTP parameters:
*
* MTP_CONFIG <reserved> <reserved> <options>
MTP_CONFIG  0  0  0x00040000
*
* Define linksets:
* MTP_LINKSET <linkset_id> <adjacent_spc> <num_links> <flags> <local_spc> <ssf>
MTP_LINKSET  0  1  2  0x0000  2  0x0008
*
* Define signaling links:
* MTP_LINK <link_id> <linkset_id> <link_ref> <slc> <board_id> <blink>
*          <stream> <timeslot> <flags>
*
*
* For SPCI4 / SPCI2S, SS7MD and SS7LD boards:
*MTP_LINK  0  0  0  0  0  0    0   16  0x0006
*MTP_LINK  1  0  1  1  1  0    0   1   0x0006
* For SS7HD boards:
*MTP_LINK  0  0  0  0  0  0-0  0   16  0x0006
*MTP_LINK  1  0  1  1  1  0-1  0   1   0x0006
*
* Define QSAAL links (SS7MD boards only):
* MTP_LINK <link_id> <linkset_id> <link_ref> <slc> <board_id> <blink>
*          <atm_stream> <vpi-vci> <flags> ATM
*
* MTP_LINK  0  0  0  0  0  0  0  5-10  0x0006  ATM
*
* Define a route for each remote signaling point:
* MTP_ROUTE <dpc> <linkset_id> <user_part_mask>
MTP_ROUTE  1  0  0x0020
*
* Define any user provided Layer 4 protocol:
* MTP_USER_PART <service_ind> <module_id>
*MTP_USER_PART  0x0a  0x2d
*
*
* ISUP parameters:
*
* Configure ISUP module:
* ISUP_CONFIG <reserved> <reserved> <user_id> <options> <num_grps> <num_ccts>
*ISUP_CONFIG  0  0  0x1d  0x0435  4  64
*
* Configure ISUP circuit groups:
* ISUP_CFG_CCTGRP <gid> <dpc> <base_cic> <base_cid> <cic_mask> <options>
*                       <user_inst> <user_id> <opc> <ssf> <variant> <options2>
*ISUP_CFG_CCTGRP  0  1  0x01  0x01  0x7fff7fff  0x001c  0  0x1d  2  0x8  0  0x00
*
*
*  TUP parameters:
* Configure TUP module:
* TUP_CONFIG <reserved> <reserved> <user_id> <options> <num_grps> <num_ccts>
*TUP_CONFIG  0  0  0x1d  0x8141  4  64
*
* Define TUP circuit groups:
* TUP_CFG_CCTGRP <gid> <dpc> <base_cic> <base_cid> <cic_mask> <options>
```

```
*                      <user_inst> <user_id> <opc> <ssf>
*TUP_CFG_CCTGRP  0  1  0x01  0x01  0x7fff7fff  0x0030  0  0x1d  2  0x08
*
*
* SCCP parameters:
*
* Configure SCCP module:
* SCCP_CONFIG <local_spc> <ssf> <options> [<send_uis>]
* SCCP_CONFIG 123 8 0
*
* Configure SCCP Sub-System Resource
* SCCP_SSR <ssr_id> RSP <remote_spc> <flags> <pc_mask>
* SCCP_SSR 1 RSP 1236 0
*
* SCCP_SSR <ssr_id> LSS <local_ssn> <module_id> <flags> <protocol>
* SCCP_SSR 2 LSS 0x07 0x0d 1 TCAP
*
* SCCP_SSR <ssr_id> RSS <remote_spc> <remote_ssn> <flags>
* SCCP_SSR 3 RSS 1236 0x67 0
*
* SCCP Concerned Sub-System Resource
* SCCP_CONC_SSR <id> <cssr_id> <ssr_id>
* SCCP_CONC_SSR 1 2 3
*
* Configure SCCP Trace
* SCCP_TRACE <op_evt_mask> <ip_evt_mask> <non_prim_mask>
* SCCP_TRACE 0x1 0x1 0x1
*
* Define Global Title Pattern
* SCCP_GTT_PATTERN <pattern_id> <addr_indicator> <pc> <ssn> <global_title>
[<gtai_pattern>]
* SCCP_GTT_PATTERN 5 0x10 0x0000 0 0x001104 44/+
*
* Define Global Title Address
* SCCP_GTT_ADDRESS <address_id> <addr_indicator> <pc> <ssn> <global_title>
[<gtai_replacement>]
* SCCP_GTT_ADDRESS 9 0x11 0x1234 0 0x001104 0-/-
*
* Add Entry in GTT Table
* SCCP_GTT <pattern_id> <mask> <primary_address_id> [<backup_address_id>]
* SCCP_GTT 5 R-/K 9
*
*
* TCAP parameters:
*
* Configure TCAP
* TCAP_CONFIG <base_ogdlg_id> <nog_dialogues> <base_icdlg_id> <nic_dialogues> <options>
<dlg_hunt>
*              [<addr_format>]
* TCAP_CONFIG 0x0000 8192 0x8000 8192 0x0000 0
*
* Define TCAP circuit groups:
* TCAP_CFG_DGRP <gid> <base_ogdlg_id> <nog_dialogues> <base_icdlg_id> <nic_dialogues>
<options>
*                <reserved>
* TCAP_CFG_DGRP 0 0x0000 1024 0x8000 1024 0 0
*
* Configure TCAP Trace
```

```
* TCAP_TRACE <op_evt_mask> <ip_evt_mask> <non_prim_mask>
* TCAP_TRACE 0x7 0xf 0x0
*
*
* MAP parameters:
*
* Configure MAP
* MAP_CONFIG <options>
* MAP_CONFIG 2
*
* Configure MAP Trace
* MAP_TRACE <op_evt_mask> <ip_evt_mask> <non_prim_mask>
* MAP_TRACE 0xf 0xf 0x4
*
*
* INAP parameters:
*
* Configure INAP
* INAP_CONFIG <options>
* INAP_CONFIG 2
*
* Configure INAP Functional Entities
* INAP_FE <fe_ref> <options> <sccp_address>
* INAP_FE 0x00000007 0x0000000f 0x00000000
*
* Configure INAP Application Context
* INAP_AC <ac_ref> <ac>
* INAP_AC 0x00 0xa109060704000101010000
*
* Configure INAP Trace
* INAP_TRACE <op_evt_mask> <ip_evt_mask> <non_prim_mask>
* INAP_TRACE 0xf 0xf 0x7f
*
*
* IS41 parameters:
*
* Configure IS41 TRACE
* IS41_TRACE <op_evt_mask> <ip_evt_mask> <non_prim_mask>
* IS41_TRACE 0xf 0xf 0xff
*
*******************************************************************************
```

9.3 Example M3UA ASP Config.txt – Multiple SG

```
*
* Example config.txt for the SIGTRAN Host Package.
*
* This example has a single LAS connecting to two SGs with multiple
* associations. Each route is added to both SGs so traffic will be load
* shared between them if availability allows (Load sharing across 2 SGs max).
* Traffic to each SG will also be load shared across two associations.
* (Load share works across max 4 associations to an SG)

* Local IP Address Configuration
CNSYS:IPADDR=192.168.0.1,IPADDR2=192.168.1.1,DAUD=Y;

* Local AS configuration
SNAPI:LAS=1,OPC=104,TRMD=LS;

* SCTP Association configuration to Remote SG
SNSLI:SNLINK=1,IPADDR=192.168.0.2,IPADDR2=192.168.1.2,SNTYPE=M3UA,SNEND=C,SG=1;
SNSLI:SNLINK=2,IPADDR=192.168.0.3,IPADDR2=192.168.1.3,SNTYPE=M3UA,SNEND=C,SG=1;
SNSLI:SNLINK=3,IPADDR=192.168.0.4,IPADDR2=192.168.1.4,SNTYPE=M3UA,SNEND=C,SG=2;
SNSLI:SNLINK=4,IPADDR=192.168.0.5,IPADDR2=192.168.1.5,SNTYPE=M3UA,SNEND=C,SG=2;

* Define routes
SNRTI:SNRT=1,DPC=101;
SNRTI:SNRT=2,DPC=102;

* Add routes to SG
SNRLI:SNRL=1,SNRT=1,SG=1;
SNRLI:SNRL=2,SNRT=1,SG=2;
SNRLI:SNRL=3,SNRT=2,SG=1;
SNRLI:SNRL=4,SNRT=2,SG=2;

* Bind LAS to SG
SNLBI:SNLB=1,LAS=1,SG=1;

* ISUP parameters for Default NC0/LAS1:
* Configure ISUP module:
* ISUP_CONFIG <reserved> <reserved> <user_id> <options> <num_grps> <num_ccts>
ISUP_CONFIG 0 0 0x1d 0x0435 4 64
*
* Configure ISUP circuit groups:
* ISUP_CFG_CCTGRP <gid> <dpc> <base_cic> <base_cid> <cic_mask> <options>
*                 <user_inst> <user_id> <opc> <ssf> <variant> <options2>
ISUP_CFG_CCTGRP 0 101 0x01 0x01 0x7fff7fff 0x001c 0 0x1d 104 0x08 0 0x00
ISUP_CFG_CCTGRP 1 102 0x21 0x21 0x7fff7fff 0x001c 0 0x1d 104 0x08 0 0x00
```

9.4 Example M3UA IPSP Config.txt – Multiple RAS

```
*
* Example config.txt for the SIGTRAN Host Package.
*
* This example has a single LAS connecting to multiple RAS.
* The single LAS has an association to each RAS so there are no shared
* associations. Each RAS has an optional RC defined by SNRAI.

* Local IP Address Configuration
CNSYS:IPADDR=192.168.0.1;

* Local AS configuration
SNAPI:LAS=1,OPC=104;

* SCTP Association configuration to Remote IPSP
SNSLI:SNLINK=1,IPADDR=192.168.0.2,HPORT=2905,PPORT=2905,SNTYPE=M3UA,SNEND=S;
SNSLI:SNLINK=2,IPADDR=192.168.0.3,HPORT=2906,PPORT=2906,SNTYPE=M3UA,SNEND=S;

* Define Remote AS
SNRAI:RAS=1,DPC=101,RC=1;
SNRAI:RAS=2,DPC=102,RC=2;

* Add Remote AS to Association
SNALI:SNAL=1,RAS=1,SNLINK=1;
SNALI:SNAL=2,RAS=2,SNLINK=2;

* Bind LAS to RAS
SNLBI:SNLB=1,LAS=1,RAS=1;
SNLBI:SNLB=2,LAS=1,RAS=2;

* ISUP parameters for Default NC0/LAS1:
* Configure ISUP module:
* ISUP_CONFIG <reserved> <reserved> <user_id> <options> <num_grps> <num_ccts>
ISUP_CONFIG 0 0 0x3d 0x0435 32 1024
*
* Configure ISUP circuit groups:
* ISUP_CFG_CCTGRP <gid> <dpc> <base_cic> <base_cid> <cic_mask> <options>
*                 <user_inst> <user_id> <opc> <ssf> <variant> <options2>
ISUP_CFG_CCTGRP 0 101 0x01 0x01 0x7fff7fff 0x001c 0 0x3d 104 0x00 0 0x00
ISUP_CFG_CCTGRP 1 102 0x21 0x21 0x7fff7fff 0x001c 0 0x3d 104 0x00 0 0x00
```

9.5 Example M3UA ASP Config.txt – Multiple LAS

```
*
* Example config.txt for the SIGTRAN Host Package.
*
* This example has multiple LAS connecting to a single SG with LAS1 handling ISUP
* traffic and LAS2 handling SCCP traffic.
* In this example the multiple LAS share a single association to the SG.
* This requires the RC parameter to be configured with SNLBI for each LAS-SG
* relationship, to identify the traffic on the shared association.
* Routes must be configured with each LAS specified (default is LAS1).
* ISUP config is in this file since it is for the default NC0 (LAS1).
* SCCP config must be external to this file for NC1 (LAS2).

* Local IP Address Configuration
CNSYS:IPADDR=192.168.0.1,DAUD=Y;

* Local AS configuration
SNAPI:LAS=1,OPC=104;
SNAPI:LAS=2,OPC=105;

* SCTP Association configuration to Remote SG
SNSLI:SNLINK=1,IPADDR=192.168.0.2,SNTYPE=M3UA,SNEND=C,SG=1;

* Define routes
SNRTI:SNRT=1,DPC=101,LAS=1;
SNRTI:SNRT=2,DPC=102,LAS=1;
SNRTI:SNRT=3,DPC=101,LAS=2;
SNRTI:SNRT=4,DPC=103,LAS=2;

* Add routes to SG
SNRLI:SNRL=1,SNRT=1,SG=1;
SNRLI:SNRL=2,SNRT=2,SG=1;
SNRLI:SNRL=3,SNRT=3,SG=1;
SNRLI:SNRL=4,SNRT=4,SG=1;

* Bind LAS to SG
SNLBI:SNLB=1,LAS=1,SG=1,RC=7;
SNLBI:SNLB=2,LAS=2,SG=1,RC=9;

* ISUP parameters for Default NC0/LAS1:
* Configure ISUP module:
* ISUP_CONFIG <reserved> <reserved> <user_id> <options> <num_grps> <num_ccts>
ISUP_CONFIG 0 0 0x1d 0x0435 4 64
*
* Configure ISUP circuit groups:
* ISUP_CFG_CCTGRP <gid> <dpc> <base_cic> <base_cid> <cic_mask> <options>
*                 <user_inst> <user_id> <opc> <ssf> <variant> <options2>
ISUP_CFG_CCTGRP 0 101 0x01 0x01 0x7fff7fff 0x001c 0 0x1d 104 0x08 0 0x00
ISUP_CFG_CCTGRP 1 102 0x21 0x21 0x7fff7fff 0x001c 0 0x1d 104 0x08 0 0x00

* External module for SCCP (SI=0x03) for NC1/LAS2:
* MTP_USER_PART [NC] <service_ind> <module_id>
MTP_USER_PART    NC1     0x03          0x2d
```

9.6 Example M3UA IPSP (Client) Config.txt

```
*
* Example config.txt for the SIGTRAN Host Package.
*
* This example has multiple LAS connecting to a single RAS.
* It can be used in conjunction with the following example as the peer.
* The multiple LAS share a single association to the RAS. This requires the RC
* parameter to be configured for each LAS-RAS relationship, to identify the traffic on
* the shared association. RC is defined by SNLBI rather than SNRAI.

* Local IP Address Configuration
CNSYS:IPADDR=192.168.0.1;

* Local AS configuration
SNAPI:LAS=1,OPC=101;
SNAPI:LAS=2,OPC=102;

* SCTP Association configuration to Remote IPSP
SNSLI:SNLINK=1,IPADDR=192.168.0.2,HPORT=2905,PPORT=2905,SNTYPE=M3UA,SNEND=C;

* Define Remote AS
SNRAI:RAS=1,DPC=103;

* Add Remote AS to Association
SNALI:SNAL=1,RAS=1,SNLINK=1;

* Bind LAS to RAS
SNLBI:SNLB=1,LAS=1,RAS=1,RC=5;
SNLBI:SNLB=2,LAS=2,RAS=1,RC=6;

* Connect to external user parts for NC0/LAS1 and NC1/LAS2
* for testing use s7_log (0xef) to display traffic
* MTP_USER_PART [NC] <service_ind> <module_id>
MTP_USER_PART    NC0     0x03          0xef
MTP_USER_PART    NC1     0x05          0xef
```

9.7 Example M3UA IPSP (Server) Config.txt

```
*
* Example config.txt for the SIGTRAN Host Package.
*
* This example has a single LAS connecting to multiple RAS.
* It can be used in conjunction with the preceding example as the peer.
* The LAS shares a single association to the multiple RAS. This requires the RC
* parameter to be configured for each LAS-RAS relationship, to identify the traffic on
* the shared association. RC is defined by SNLBI rather than SNRAI.

* Local IP Address Configuration
CNSYS:IPADDR=192.168.0.2;

* Local AS configuration
SNAPI:LAS=1,OPC=103;

* SCTP Association configuration to Remote IPSP
SNSLI:SNLINK=1,IPADDR=192.168.0.1,HPORT=2905,PPORT=2905,SNTYPE=M3UA,SNEND=S;

* Define Remote AS
SNRAI:RAS=1,DPC=101;
SNRAI:RAS=2,DPC=102;

* Add Remote AS to Association
SNALI:SNAL=1,RAS=1,SNLINK=1;
SNALI:SNAL=2,RAS=2,SNLINK=1;

* Bind LAS to RAS
SNLBI:SNLB=1,LAS=1,RAS=1,RC=5;
SNLBI:SNLB=2,LAS=1,RAS=2,RC=6;

* Connect to external user parts for NC0/LAS1
* for testing use s7_log (0xef) to display traffic
* MTP_USER_PART [NC] <service_ind> <module_id>
MTP_USER_PART    NC0     0x03          0xef
MTP_USER_PART    NC0     0x05          0xef
```

9.8 Example M2PA Configuration

```
*
* Example config.txt for the SIGTRAN Host Package.
*
* Edit this file to reflect your configuration.
*
* SYSTEM Parameters
CNSYS:IPADDR=192.168.0.1,PER=0;
SNSLI:SNLINK=1,IPADDR=192.168.0.2,SNEND=C,SNTYPE=M2PA,M2PA=1, HPORT=3565,PPORT=3565;
SNSLI:SNLINK=2,IPADDR=192.168.0.2,SNEND=C,SNTYPE=M2PA,M2PA=2, HPORT=3566,PPORT=3566;
SNSLI:SNLINK=3,IPADDR=192.168.0.2,SNEND=C,SNTYPE=M2PA,M2PA=3, HPORT=3567,PPORT=3567;
SNSLI:SNLINK=4,IPADDR=192.168.0.2,SNEND=C,SNTYPE=M2PA,M2PA=4, HPORT=3568,PPORT=3568;
SNSLI:SNLINK=5,IPADDR=192.168.0.2,SNEND=C,SNTYPE=M2PA,M2PA=5, HPORT=3569,PPORT=3569;
SNSLI:SNLINK=6,IPADDR=192.168.0.2,SNEND=C,SNTYPE=M2PA,M2PA=6, HPORT=3570,PPORT=3570;
SNSLI:SNLINK=7,IPADDR=192.168.0.2,SNEND=C,SNTYPE=M2PA,M2PA=7, HPORT=3571,PPORT=3571;
SNSLI:SNLINK=8,IPADDR=192.168.0.2,SNEND=C,SNTYPE=M2PA,M2PA=8, HPORT=3572,PPORT=3572;

MTP_CONFIG  0  0  0x00000000
*
* Define linksets:
* MTP_LINKSET <linkset_id> <adjacent_spc> <num_links> <flags> <local_spc> <ssf>
MTP_LINKSET  0  10  8 0x0000  100  0x00
*
* Define signaling links:
* MTP_LINK <link_id> <linkset_id> <link_ref> <slc> <board_id> <blink>
*          <stream> <timeslot> <flags>
MTP_LINK  0  0  0  0  0  1   0   0  0x80000006
MTP_LINK  1  0  1  1  0  2   0   0  0x80000006
MTP_LINK  2  0  2  2  0  3   0   0  0x80000006
MTP_LINK  3  0  3  3  0  4   0   0  0x80000006
MTP_LINK  4  0  4  4  0  5   0   0  0x80000006
MTP_LINK  5  0  5  5  0  6   0   0  0x80000006
MTP_LINK  6  0  6  6  0  7   0   0  0x80000006
MTP_LINK  7  0  7  7  0  8   0   0  0x80000006
*
* MTP_ROUTE <dpc> <linkset_id> <user_part_mask>
MTP_ROUTE  10  0  0x0028
*
MTP_USER_PART  0x0a  0x1d
```

9.9 Example GTT Configuration

```
*
*  Specific Address to PC + SSN
*  This example translates a received specific Global Title address             *
(09876543210) into a combination of Point Code (138) and SSN (3).
*
* SCCP_GTT_PATTERN <pattern_id> <addr_indicator> <pc>     <ssn> <global_title>
[<gtai_pattern>]
SCCP_GTT_PATTERN    1   0x10    0        0     0x001104       09876543210
SCCP_GTT_ADDRESS    1   0x03    138      8     0x00           -
SCCP_GTT            1                                       R           1
*
* Match a 7 digit number starting "123", followed by any three digits, then "7".
* change the first digits to "333" keep the next three digits from the called-  * party
address and change the fourth digit to "4", and add a PC & SSN.
*
SCCP_GTT_PATTERN    2   0x10    0        0     0x001104       123/???/7
SCCP_GTT_ADDRESS    2   0x11    11       0     0x001104       333/---/4
SCCP_GTT            2                                       R--/K--/R   2

* A Matching Prefix to PC + SSN
*  This example translates any global title address matching a pattern          *
consisting of a prefix (441425) following by a suffix of any digits and any   * length
into the digits minus the prefix, and adding a PC (238) and SSN (3).
*
SCCP_GTT_PATTERN    3   0x10    0        0     0x001104       441425/+
SCCP_GTT_ADDRESS    3   0x13    238      3     0x001104       -/-
SCCP_GTT            3                                       R/K         3

* A Matching Prefix to PC + SSN
*  Ignoring any preceding digits, match "1425" followed by any six digits.      *
Remove any digits preceding the “1425”, keeping the final six digits from the * Input
GTAI. Add a PC & SSN.
*
SCCP_GTT_PATTERN    4   0x10    0        0     0x001104       +/1425/??????
SCCP_GTT_ADDRESS    4   0x13    128      9     0x001104       -/-/-
SCCP_GTT            4                                       R/K/K       4

* Adding a PC + SSN to any GTAI
* This example matches any GTAI Digits and adds a Point Code and SSN, retaining * any
GTAI digits.
*
SCCP_GTT_PATTERN    5   0x10    0x0000   0x03  0x001204       +/-
SCCP_GTT_ADDRESS    5   0x53    0x3FFF   0x08  0x001204       -/e
SCCP_GTT            5                                       K/R         5
```

9.10 Example Configuration of an ATM Terminated Link

Example configuration for a terminated ATM link.

```
*
* Example Protocol Configuration File (config.txt) for use with
* Dialogic(R) DSI SS7MD Network Interface Boards.
*
* This file needs to be modified to suit individual circumstances.
* Refer to the relevant Programmer's Manuals for further details.
*
* SS7_BOARD   <board_id> <board_type> <flags> <code_file> <run_mode>
SS7_BOARD     0          SS7MD        0x0000  ./DC/ss7.dc6     ATM
*
* LIU_CONFIG <board_id> <liu_id> <liu_type> <line_code> <frame_format>          *
<crc_mode> [<build_out>]
LIU_CONFIG   0          0        5           1           1              1        0
*
* ATM_CONFIG  <options> <num_streams>
ATM_CONFIG    0x0000    4
*
*
* ATM_STREAM   <id> <board_id> <cellstream_id> <liu_id> <options> <ima_frame_len> <max_
frame_len> <def_vpi> <def_vci> <timeslot>
ATM_STREAM     3  0  1  0  0x01  0 280  12  10  0xfffefffe
*
MTP_CONFIG   <reserved1> <reserved2> <options>
MTP_CONFIG   0           0           0x00040000
*
* MTP_LINKSET <linkset_id> <adjacent_spc> <num_links> <flags> <local_spc> <ssf>
MTP_LINKSET   0            1              1           0x0000  2           0x08
*
* MTP_LINK <link_id> <linkset_id> <link_ref> <slc> <board_id> <blink>           *
<atm_stream> <vpi-vci > <flags> [<data_rate>]
MTP_LINK   0  0  0  0  0  0 3  8-100  0x0006 ATM
*
* MTP_ROUTE <dpc> <linkset_id> <user_part_mask>
MTP_ROUTE   1     0            0x0020
```

9.11 Example Diameter Configuration

The DSI Development Pack includes example applications for the DSI Diameter Stack. A sample configuration file for use with the DTU example application is shown below. This file and the equivalent example configuration file for the partner example application DTR are included in the UPD/RUN sub-directory of the Development Package.

```
*****************************************************************************
*
* Example protocol configuration file for the Dialogic(R) DSI Diameter Stack
*
* Diameter - DTU Basic configuration:
*
*****************************************************************************
*
* Local IP Address Configuration
*
CNSYS:IPADDR=192.168.0.2;
*
* Set per-module options for Diameter
*
* CNOPS:MODULE=DMR,MOD_ID=0x74;
*
* Set module-specific options for Diameter
*
DMSYI:BASEOGD=0x0000,BASEICD=0x8000;
*
* Configure an SCTP association
*
SNSLI:SNLINK=1,IPADDR=192.168.0.1,HIPADDR1=192.168.0.2,SNEND=C,SNTYPE=DMR,PPORT=3868,HPORT=3868;
*
* Configure the Diameter Network Context
*
DMNCI:DMNC=0,OPTIONS=0x00000000,HOST=dmr01.lab.dialogic.com,REALM=dialogic.com,NODENAME=ExampleMME
,LABEL=London;
*
* Configure the Peer, linking the DMNC to the SNLINK
* (Options bit 0 must be set for Server operation)
*
DMPRI:DMPR=0,DMNC=0,OPTIONS=0x00000000,SNLINK=1,HOST=dmr02.lab.dialogic.com,REALM=dialogic.com,LAB
EL=Paris;
*
* Configure the Route
*
DMRTI:DMRT=0,DMNC=0,APPID=S6a,OPTIONS=0x00000000,HOST=dmr02.lab.dialogic.com,LABEL=Primary;
*
* Identify the Peers which can be used by a Route
*
DMRLI:DMRL=0,DMPR=0,DMRT=0;
* Configure the Application which uses the DMNC
*
DMAPI:DMAP=0,OPTIONS=0x00000000,MOD_ID=0x1d,MOD_INST=0,DMNC=0,APPID=S6a,VENDORID=0,LABEL=Billing;
DMAPI:DMAP=1,OPTIONS=0x00000000,MOD_ID=0x1d,MOD_INST=0,DMNC=0,APPID=CC,VENDORID=0,LABEL=CreditCont
rol;
*
*
*****************************************************************************
```

Appendix A. Default Module Identifiers

The default module identifiers are listed in the following table. In some systems, these default values may be changed at run-time when the system is run up, so care should be taken to understand that the module identifier is not necessarily fixed to the default value.

Module identifiers with a least significant nibble set to 0x0d are reserved for user-generated applications. Although the values may also be used in example applications supplied by Dialogic.

Module identifiers with a least significant nibble set to 0x0c are reserved entirely for user-generated applications. These 16 module identifiers will not be used in any Dialogic® DSI Components and are therefore available for use by the user in custom applications.

Table 10. Default module identifier values

Value	Mnemonic	Description
0x00	TIM_MOD_ID	Timer module
0x10	MVD_TASK_ID	Physical switch & clock driver (per-board)
0x20	SSD_TASK_ID	Physical board interface module
0x80	DVR_SP0_TASK_ID	Driver for SP0
0x90	DVR_SP1_TASK_ID	Driver for SP1
0xb0	RSI_MOD_ID	RSI socket based interface
0xe0	DVR_SP2_TASK_ID	Driver for SP2
0xf0	DVR_SP3_TASK_ID	Driver for SP3
0x21	CONG_TASK_ID	Congestion module
0x31	ATM_TASK_ID	ATM Module
0x41	QSL_TASK_ID	Q.SAAL Module
0x61	DVR_ALT_TASK_ID	SS7MD Signaling Driver Module
0x71	SS7_TASK_ID	MTP2 protocol module
0x74	DMR_TASK_ID	DMR Diameter Module
0x81	SS7_SP0_TASK_ID	MTP2 for SP0 (SS7HD only)
0x91	SS7_SP1_TASK_ID	MTP2 for SP1 (SS7HD only)
0xb1	MST_TASK_ID	SIGTRAN Monitor task
0xc1	M2P_TASK_ID	M2PA protocol module
0xe1	SS7_SP2_TASK_ID	MTP2 for SP2 (SS7HD only)
0xf1	SS7_SP3_TASK_ID	MTP2 for SP3 (SS7HD only)
0x22	MTP_TASK_ID	MTP3 protocol module
0x32	RMM_TASK_ID	RMM module
0xd2	M3UA_TASK_ID	M3UA protocol module
0x23	ISP_TASK_ID	ISUP protocol module

Value	Mnemonic	Description
0x33	SCP_TASK_ID	SCCP protocol module
0xc3	SUA_TASK_ID	SUA protocol module
0x14	TCP_TASK_ID	TCAP protocol module
0x15	MAP_TASK_ID	MAP protocol module
0x25	IS41_TASK_ID	IS41 protocol module
0x35	INAP_TASK_ID	INAP protocol module
0x4a	TUP_TASK_ID	TUP protocol module
0x0d	APP0_TASK_ID	User's application module
0x1d	APP1_TASK_ID	User's application module
0x2d	APP2_TASK_ID	User's application module
0x3d . . 0xcd.		User's application module
0xdd	APP13_TASK_ID	User's application module
0xed	APP14_TASK_ID	User's application module
0xfd	APP15_TASK_ID	User's application module
0x8e	MGMT_TASK_ID	General management module
0xce	MGMT_SP0_TASK_ID	Management Module for SP0
0xde	MGMT_SP1_TASK_ID	Management Module for SP1
0xee	MGMT_SP2_TASK_ID	Management Module for SP2
0xfe	MGMT_SP3_TASK_ID	Management Module for SP3
0xcf		s7_mgt - Management/config task
0xdf	SIU_MGT_TASK_ID	Internal SIU use
0xef	REM_API_ID	Remote (users) application
0xff		Invalid module_id - do not use!

Appendix B. Values reserved for Custom Use

In some cases, users may wish to add their own modules and messages to a system. To this end, a range of module identifiers and message types have been reserved for this purpose and will not be used in Dialogic® DSI Components.

B.1 Reserved module identifiers

All module_id values containing 0xc as the least significant nibble are reserved for use in user-generated applications.

B.2 Reserved message types

A total of 1024 message types are reserved exclusively for use in the user's own applications.

The reserved message types are of the following 4 formats, where the nibbles identified by a question mark can be set to any value:

0x?cc?
0x?cd?
0x?ce?
0x?cf?

For example, the message types 0x1cc1, 0x2cd2, 0x3ce3 and 0x4cf4 are reserved for use in the user's applications.

Appendix C. GCTLIB Javadoc

This appendix documents the Java class library provided for access to the message passing environment. See section 6.3 Java Inter-Process Communications for further details.

C.1 com.dialogic.signaling.gct - Class BBUtil

```
java.lang.Object
  └ com.dialogic.signaling.gct.BBUtil
```

```
public class BBUtil
extends java.lang.Object
```

BBUtil - allows access to unsigned values within a ByteBuffer

---

Constructor Summary

BBUtil()

Method Summary

static int	**getU16**(java.nio.ByteBuffer byteBuffer)
static int	**getU16**(java.nio.ByteBuffer byteBuffer, int offset)
static int	**getU24**(java.nio.ByteBuffer byteBuffer)
static int	**getU24**(java.nio.ByteBuffer byteBuffer, int offset)
static long	**getU32**(java.nio.ByteBuffer byteBuffer)
static long	**getU32**(java.nio.ByteBuffer byteBuffer, int offset)
static short	**getU8**(java.nio.ByteBuffer byteBuffer)
static short	**getU8**(java.nio.ByteBuffer byteBuffer, int offset)
static void	**putU16**(java.nio.ByteBuffer byteBuffer, int value)

static void	**putU16**(java.nio.ByteBuffer byteBuffer, int offset, int value)
static void	**putU24**(java.nio.ByteBuffer byteBuffer, int value)
static void	**putU24**(java.nio.ByteBuffer byteBuffer, int offset, int value)
static void	**putU32**(java.nio.ByteBuffer byteBuffer, int offset, long value)
static void	**putU32**(java.nio.ByteBuffer byteBuffer, long value)
static void	**putU8**(java.nio.ByteBuffer byteBuffer, int value)
static void	**putU8**(java.nio.ByteBuffer byteBuffer, int offset, int value)

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

BBUtil

public **BBUtil**()

Method Detail

getU8

public static short **getU8**(java.nio.ByteBuffer byteBuffer)

---

putU8

public static void **putU8**(java.nio.ByteBuffer byteBuffer,
int value)

---

getU8

public static short **getU8**(java.nio.ByteBuffer byteBuffer,
int offset)

---

putU8

```
public static void putU8(java.nio.ByteBuffer byteBuffer,
                         int offset,
                         int value)
```

---

getU16

```
public static int getU16(java.nio.ByteBuffer byteBuffer)
```

---

putU16

```
public static void putU16(java.nio.ByteBuffer byteBuffer,
                          int value)
```

---

getU16

```
public static int getU16(java.nio.ByteBuffer byteBuffer,
                         int offset)
```

---

putU16

```
public static void putU16(java.nio.ByteBuffer byteBuffer,
                          int offset,
                          int value)
```

---

getU32

```
public static long getU32(java.nio.ByteBuffer byteBuffer)
```

---

putU32

```
public static void putU32(java.nio.ByteBuffer byteBuffer,
                          long value)
```

---

getU32

```
public static long getU32(java.nio.ByteBuffer byteBuffer,
                          int offset)
```

---

putU32

```
public static void putU32(java.nio.ByteBuffer byteBuffer,
                          int offset,
                          long value)
```

---

getU24

```
public static int getU24(java.nio.ByteBuffer byteBuffer)
```

---

putU24

```
public static void putU24(java.nio.ByteBuffer byteBuffer,
                          int value)
```

---

getU24

```
public static int getU24(java.nio.ByteBuffer byteBuffer,
                         int offset)
```

---

putU24

```
public static void putU24(java.nio.ByteBuffer byteBuffer,
                          int offset,
                          int value)
```

C.2 com.dialogic.signaling.gct - Class GctException

```
java.lang.Object
  └ java.lang.Throwable
      └ java.lang.Exception
          └ com.dialogic.signaling.gct.GctException
```

All Implemented Interfaces:

java.io.Serializable

```
public class GctException
extends java.lang.Exception
```

See Also:

Serialized Form

Constructor Summary

GctException(java.lang.String message)

Method Summary

Methods inherited from class java.lang.Throwable
fillInStackTrace, getCause, getLocalizedMessage, getMessage, getStackTrace, initCause, printStackTrace, printStackTrace, printStackTrace, setStackTrace,

toString

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

GctException

public **GctException**(java.lang.String message)

C.3 com.dialogic.signaling.gct - Class GctLib

java.lang.Object
  └ **com.dialogic.signaling.gct.GctLib**

public class **GctLib**
extends java.lang.Object

Nested Class Summary

static class	**GctLib.PartitionInfo**
static class	**GctLib.StandardMsgSizes**

Field Summary

static java.lang.String	**GctLibVersionNumber**

Constructor Summary

GctLib()

Method Summary

static GctMsg	**getm**(GctLib.StandardMsgSizes size) Get a new GctMsg object.
static GctMsg	**getm**(int len)

	Get a new GctMsg object.
static GctMsg	**getm**(int type, int id, int rspReq, int len) Get a new GctMsg object.
static GctLib.PartitionInfo	**getPartitionInfo**(int partitionId) Gets information about the specified partition.
static GctMsg	**grab**(short taskId) Non blocking call to receive a new Msg on the identified taskId.
static boolean	**isPartitionCongested**(int partitionId) Determines whether the native message partition is currently congested.
static void	**link**() Establishes a link to the GCT environment
static int	**pendingMsgs**(short taskId) Returns the number of messages queued against the task.
static GctMsg	**receive**(short taskId) Blocking call waiting to receive a new Msg on the identified taskId.
static void	**relm**(GctMsg msg) Returns the underlying native Msg resource for reuse.
static void	**send**(GctMsg msg) Sends the Msg.
static void	**send**(short taskId, GctMsg msg) Sends the Msg to the identified taskId.
static void	**unlink**() Closes a link to the GCT environment

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Field Detail

GctLibVersionNumber

public static final java.lang.String **GctLibVersionNumber**

See Also:

Constant Field Values

Constructor Detail

GctLib

```
public GctLib()
```

Method Detail

getm

```
public static GctMsg getm(int type,
                          int id,
                          int rspReq,
                          int len)
                   throws GctException
```

Get a new GctMsg object. The returned GctMsg wraps a native GCT message from the nativeGCT message passing environment.

Parameters:

`type` - Set the GctMsg type field to this value

`id` - Set the GctMsg id field to this value

`rspReq` - Set the GctMsg rspReq to this value

`len` - Get a message of at least this length and set the GctMsg len field to this value

Returns:

A new GctMsg message wrapping a native Msg

Throws:

`GctException` - If a native Msg could not be allocated

---

getm

```
public static GctMsg getm(int len)
                   throws GctException
```

Get a new GctMsg object. The returned GctMsg wraps a native GCT message from the native message passing environment. The message parameter area will be at least 'len' bytes long.

Parameters:

`len` - Get a message of at least this length and set the GctMsg len field to this value

Returns:

A new GctMsg message wrapping a native Msg

Throws:

`GctException` - If a native Msg could not be allocated

---

getm

```
public static GctMsg getm(GctLib.StandardMsgSizes size)
                   throws GctException
```

Get a new GctMsg object. The returned GctMsg wraps a native GCT message from the GCT message passing environment. The message parameter area will be at least 320 bytes long.

Returns:

A new GctMsg message wrapping a native Msg

Throws:

`GctException` - If a native Msg could not be allocated

---

relm

```
public static void relm(GctMsg msg)
                 throws GctException
```

Returns the underlying native Msg resource for reuse. Note: the GctMsg object may be separately disposed of.

Parameters:

`msg` - The GctMsg to be released

Throws:

`GctException` - If the underlying native Msg is null or failed to release Msg

---

send

```
public static void send(short taskId,
                        GctMsg msg)
                 throws GctException
```

Sends the Msg to the identified taskId. Note: This actually sends the underlying native Gct Msg.

Parameters:

`taskId` - The taskId or moduleId to send the Msg to

`msg` - The Msg to send

Throws:

`GctException` - If the native Msg is null or failed to send

---

send

```
public static void send(GctMsg msg)
                 throws GctException
```

Sends the Msg. The destination taskId is that within the message header Note: This actually sends the underlying native Gct Msg.

Parameters:

`msg` - The Msg to send

Throws:

`GctException` - If the native Msg is null or failed to send

---

receive

```
public static GctMsg receive(short taskId)
```

Blocking call waiting to receive a new Msg on the identified taskId. Note: This waits for a native Gct Msg to be received on the given taskId and wraps it in a GctMsg object.

Parameters:
`taskId` - Task Id to wait for a message on.
Returns:
GctMsg

---

grab

```
public static GctMsg grab(short taskId)
```

Non blocking call to receive a new Msg on the identified taskId. If there are no messages waiting then this function returns immediately. Note: This wraps the native Msg in a GctMsg object.
Parameters:
`taskId` - Task Id to wait for a message on.
Returns:
GctMsg or null

---

link

```
public static void link()
```

Establishes a link to the GCT environment

---

unlink

```
public static void unlink()
```

Closes a link to the GCT environment

---

isPartitionCongested

```
public static boolean isPartitionCongested(int partitionId)
```

Determines whether the native message partition is currently congested.
Parameters:
`partitionId` - The message partition to check
Returns:
True if congested, otherwise false.

---

getPartitionInfo

```
public static GctLib.PartitionInfo getPartitionInfo(int partitionId)
                                                    throws GctException
```

Gets information about the specified partition.
Parameters:
`partitionId` - The native message partition for which information is requested.
Returns:
PartitionInfo instance

Throws:
`GctException`

---

pendingMsgs

`public static int` **`pendingMsgs`**`(short taskId)`

Returns the number of messages queued against the task.

Parameters:

`taskId` - The task Id for which the number of pending messages is requested.

Returns:

The number of pending messages.

C.4 com.dialogic.signaling.gct - Class GctLib.PartitionInfo

`java.lang.Object`
`└` **`com.dialogic.signaling.gct.GctLib.PartitionInfo`**

Enclosing class:

GctLib

`public static class` **`GctLib.PartitionInfo`**
`extends java.lang.Object`

Field Summary

`boolean`	**`congStatus`**
`java.lang.Integer`	**`numMsgs`**
`java.lang.Integer`	**`paramSize`**
`java.lang.Integer`	**`partitionId`**

Method Summary

Methods inherited from class java.lang.Object
`equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Field Detail

partitionId

```
public java.lang.Integer partitionId
```

---

numMsgs

```
public java.lang.Integer numMsgs
```

---

paramSize

```
public java.lang.Integer paramSize
```

---

congStatus

```
public boolean congStatus
```

C.5 com.dialogic.signaling.gct - Enum GctLib.StandardMsgSizes

```
java.lang.Object
  └ java.lang.Enum<GctLib.StandardMsgSizes>
      └ com.dialogic.signaling.gct.GctLib.StandardMsgSizes
```

All Implemented Interfaces:

java.io.Serializable, java.lang.Comparable<GctLib.StandardMsgSizes>

Enclosing class:

GctLib

```
public static enum GctLib.StandardMsgSizes
extends java.lang.Enum<GctLib.StandardMsgSizes>
```

Enum Constant Summary

Bytes320
Bytes4200

Method Summary

static GctLib.StandardMsgSizes	**valueOf**(java.lang.String name) Returns the enum constant of this type with the specified name.

`static GctLib.StandardMsgSizes[]`	**`values`**`()` Returns an array containing the constants of this enum type, in the order they are declared.

Methods inherited from class java.lang.Enum
`compareTo, equals, getDeclaringClass, hashCode, name, ordinal, toString, valueOf`

Methods inherited from class java.lang.Object
`getClass, notify, notifyAll, wait, wait, wait`

Enum Constant Detail

Bytes320

`public static final GctLib.StandardMsgSizes` **`Bytes320`**

Bytes4200

`public static final GctLib.StandardMsgSizes` **`Bytes4200`**

Method Detail

values

`public static GctLib.StandardMsgSizes[]` **`values`**`()`

Returns an array containing the constants of this enum type, in the order they are declared. This method may be used to iterate over the constants as follows:

```
for (GctLib.StandardMsgSizes c : GctLib.StandardMsgSizes.values())
    System.out.println(c);
```

Returns:

an array containing the constants of this enum type, in the order they are declared

---

valueOf

`public static GctLib.StandardMsgSizes` **`valueOf`**`(java.lang.String name)`

Returns the enum constant of this type with the specified name. The string must match *exactly* an identifier used to declare an enum constant in this type. (Extraneous whitespace characters are not permitted.)

Parameters:

`name` - the name of the enum constant to be returned.

Returns:

the enum constant with the specified name

Throws:

`java.lang.IllegalArgumentException` - if this enum type has no constant with the specified name
`java.lang.NullPointerException` - if the argument is null

C.6 com.dialogic.signaling.gct - Class GctMsg

```
java.lang.Object
  └ com.dialogic.signaling.gct.GctMsg
```

All Implemented Interfaces:

IMsg

```
public class GctMsg
extends java.lang.Object
implements IMsg
```

This class wraps a native Gct Msg and implements the IMsg interface

---

Method Summary	
short	**getDst**() Get the destination field value of the message
int	**getId**() Get the id field value of the message
long	**getInstance**() Get the instance field value of the message
java.nio.ByteBuffer	**getParam**() Get parameter area of the message
boolean	**getRspReq**() Get the response request field value of the message
short	**getSrc**() Get the source field value of the message
short	**getStatus**() Get the status field value of the message
int	**getType**() Get the type field value of the message
void	**setDst**(short dst) Set the destination field value of the message
void	**setId**(int id) Set the id field value of the message

void	**setInstance**(long instance) Set the instance field value of the message
void	**setRspReq**(boolean rspReq) Set the raw response request field value of the message
void	**setSrc**(short src) Set the source field value of the message
void	**setStatus**(short status) Set the status field value of the message
void	**setType**(int type) Set the type field value of the message

Methods inherited from class java.lang.Object

equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Detail

getParam

```
public final java.nio.ByteBuffer getParam()
                                  throws GctException
```

Get parameter area of the message

Specified by:

getParam in interface IMsg

Throws:

GctException

---

getType

```
public final int getType()
                 throws GctException
```

Get the type field value of the message

Specified by:

getType in interface IMsg

Throws:

GctException

---

setType

```
public final void setType(int type)
                  throws GctException
```

Set the type field value of the message

Specified by:

setType in interface IMsg

Throws:

GctException

---

getId

```
public final int getId()
                throws GctException
```

Get the id field value of the message

Specified by:

getId in interface IMsg

Throws:

GctException

---

setId

```
public final void setId(int id)
                 throws GctException
```

Set the id field value of the message

Specified by:

setId in interface IMsg

Throws:

GctException

---

getSrc

```
public final short getSrc()
                  throws GctException
```

Get the source field value of the message

Specified by:

getSrc in interface IMsg

Throws:

GctException

---

setSrc

```
public final void setSrc(short src)
                  throws GctException
```

Set the source field value of the message

Specified by:

setSrc in interface IMsg

Throws:

GctException

---

getDst

```
public final short getDst()
                   throws GctException
```

Get the destination field value of the message

Specified by:

`getDst` in interface `IMsg`

Throws:

`GctException`

---

setDst

```
public final void setDst(short dst)
                  throws GctException
```

Set the destination field value of the message

Specified by:

`setDst` in interface `IMsg`

Throws:

`GctException`

---

getRspReq

```
public final boolean getRspReq()
                      throws GctException
```

Get the response request field value of the message

Specified by:

`getRspReq` in interface `IMsg`

Throws:

`GctException`

---

setRspReq

```
public final void setRspReq(boolean rspReq)
                    throws GctException
```

Set the raw response request field value of the message

Specified by:

`setRspReq` in interface `IMsg`

Throws:

`GctException`

---

getStatus

```
public final short getStatus()
                     throws GctException
```

Get the status field value of the message

Specified by:

`getStatus` in interface `IMsg`

Throws:

`GctException`

---

setStatus

```
public final void setStatus(short status)
                     throws GctException
```

Set the status field value of the message

Specified by:

`setStatus` in interface `IMsg`

Throws:

`GctException`

---

getInstance

```
public final long getInstance()
                       throws GctException
```

Get the instance field value of the message

Specified by:

`getInstance` in interface `IMsg`

Throws:

`GctException`

---

setInstance

```
public final void setInstance(long instance)
                       throws GctException
```

Set the instance field value of the message

Specified by:

`setInstance` in interface `IMsg`

Throws:

`GctException`

C.7 com.dialogic.signaling.gct Interface IMsg

All Known Implementing Classes:

GctMsg

```
public interface IMsg
```

Method Summary

short	**getDst**()

int	**getId**()
long	**getInstance**()
java.nio.ByteBuffer	**getParam**()
boolean	**getRspReq**()
short	**getSrc**()
short	**getStatus**()
int	**getType**()
void	**setDst**(short dst)
void	**setId**(int id)
void	**setInstance**(long instance)
void	**setRspReq**(boolean rspReq)
void	**setSrc**(short src)
void	**setStatus**(short status)
void	**setType**(int type)

Method Detail

getParam

```
java.nio.ByteBuffer getParam()
                            throws GctException
```

Throws:

GctException

getType

```
int getType()
            throws GctException
```

Throws:

GctException

---

setType

```
void setType(int type)
             throws GctException
```

Throws:

GctException

---

getId

```
int getId()
          throws GctException
```

Throws:

GctException

---

setId

```
void setId(int id)
           throws GctException
```

Throws:

GctException

---

getSrc

```
short getSrc()
             throws GctException
```

Throws:

GctException

---

setSrc

```
void setSrc(short src)
            throws GctException
```

Throws:

GctException

---

getDst

```
short getDst()
             throws GctException
```

Throws:

GctException

---

setDst

```
void setDst(short dst)
            throws GctException
```

Throws:

GctException

---

getRspReq

```
boolean getRspReq()
                  throws GctException
```

Throws:

GctException

---

setRspReq

```
void setRspReq(boolean rspReq)
               throws GctException
```

Throws:

GctException

---

getStatus

```
short getStatus()
                 throws GctException
```

Throws:

GctException

---

setStatus

```
void setStatus(short status)
               throws GctException
```

Throws:

GctException

---

getInstance

```
long getInstance()
                  throws GctException
```

Throws:

GctException

---

setInstance

```
void setInstance(long instance)
```

```
            throws GctException
```

Throws:

`GctException`