SINAP/SS7 Programmer's Guide

Part Number: R8052-17

January 2005

SINAP/SS7 Programmer's Guide

Stratus Technologies
R8052-17

Notice

The information contained in this document is subject to change without notice.

UNLESS EXPRESSLY SET FORTH IN A WRITTEN AGREEMENT SIGNED BY AN AUTHORIZED REPRESENTATIVE OF STRATUS TECHNOLOGIES, STRATUS MAKES NO WARRANTY OR REPRESENTATION OF ANY KIND WITH RESPECT TO THE INFORMATION CONTAINED HEREIN, INCLUDING WARRANTY OF MERCHANTABILITY AND FITNESS FOR A PURPOSE. Stratus Technologies assumes no responsibility or obligation of any kind for any errors contained herein or in connection with the furnishing, performance, or use of this document.

Software described in Stratus documents (a) is the property of Stratus Technologies Bermuda, Ltd. or the third party, (b) is furnished only under license, and (c) may be copied or used only as expressly permitted under the terms of the license.

Stratus documentation describes all supported features of the user interfaces and the application programming interfaces (API) developed by Stratus. Any undocumented features of these interfaces are intended solely for use by Stratus personnel and are subject to change without warning.

Manual Name: *SINAP/SS7 Prigrammer's Guide*

Part Number: R8052
Revision Number: 17
Updated for SINAP/SS7 Release Number: 14.2
Publication Date: January 2005

Stratus Technologies, Inc.
111 Powdermill Road
Maynard, Massachusetts 01754-3409

Contents

Figures

Tables

Preface

The Purpose of This Manual

The *SINAP/SS7 Programmer's Guide* documents the application programming interface (API) of the Stratus Intelligent Network Applications Platform (SINAP) SS7 product. It is intended for programmers developing applications to run on the SINAP/SS7 system. This manual assumes that readers are familiar with the Signaling System Number 7 (SS7) protocol and have C programming experience.

Audience

This manual is intended for programmers who write applications designed to run in an SS7 network. Before using this manual, you should be familiar with the SS7 protocol. You might also find a working knowledge of Integrated Services Digital Network (ISDN) User Part useful.

Revision Information

This manual has been revised with miscellaneous corrections to existing text for the SINAP/SS7 14.2 release, and the enhancement of Partial GTT support for ANSI, China, and TTC variants. In addition, documentation bugs were fixed.

Manual Organization

This manual contains six chapters and four appendixes.

- Chapter 1, "Introduction,"describes the SINAP/SS7 product and defines the terminology used in this guide.
- Chapter 2, "Application Programming Interface (API),"provides an overview of the SINAP/SS7 API and also describes each of its components: CASL functions, structures, include files, and primitives.
- Chapter 3, "Application Design and Development,"describes the process of developing SINAP/SS7 applications.
- Chapter 4, "Application Testing, Debugging, and Troubleshooting,"describes how to test and debug SINAP/SS7 applications.
- Chapter 5, "Sample Applications," briefly describes the sample applications provided with the SINAP/SS7 software.
- Chapter 6, "CASL Function Calls,"provides reference information about each of the functions in the CASL library.

- Appendix A, “SINAP/SS7 MML Command Summary,” provides a summary of the SINAP/SS7 Man-Machine Language (MML) commands, which are used for defining and managing a SINAP/SS7 configuration.
- Appendix B, “SINAP/SS7 Environment Variables,” lists and describes the environment variables for the SINAP/SS7 system and how to define them.
- Appendix C, “CASL Error Messages,” lists and describes the various types of error messages that might be returned by CASL functions.

Notation Conventions

This manual uses the following notation conventions.

- `Monospace` represents text that would appear on your display screen (such as commands, functions, code fragments, and names of files and directories). For example:

  The alternative format for `change-remssn` is `change-remssn`.

- *`Monospace italic`* represents terms to be replaced by literal values. In the following example, the user must replace the monospace-italic term with a literal value. For example:

  The `nmtr` program has the following syntax (where *`filename`* is the name of the file to be converted).

- **`Monospace bold`** represents user input in examples and figures that contain both user input and system output (which appears in monospace). For example:

  Issue the following MML command to create the FOPC to be used in place of the MTP routing label’s OPC (where *`network-cluster-member`* defines the OPC).

  `CREATE-FOPC:FOPC=<`*`network-cluster-member`*`>;`

- *Italics* introduces or defines new terms. For example:

  The *Terminal Handler* accepts commands in Man-Machine Language (MML).

- **Boldface** emphasizes words in text. For example:

  You **must** create a link set before you provision its member links.

- When you need to press a key on the keyboard to perform an action, the keys are indicated in SMALL CAPS, for example:

  Press CTRL-P to go to the next page or RETURN to exit the screen.

NOTE

There is an implied pressing of RETURN at the end of each command and menu response that you enter.

- The dollar sign ($) and the number sign (#) are standard default prompt signs that have a specific meaning at the UNIX prompt. Although a prompt is sometimes shown at the beginning of a command line as it would appear on the screen, you do not type it.
  - $ indicates you are logged into a user account and are subject to certain access limitations.
  - # indicates you are logged into the system administrator account and have *superuser* access. Users of this account are referred to as `root`.The # prompt sign used in an example indicates the command can only be issued by `root`.
- When the full path name of a command appears in an example (for example, `/etc/fsck`), you must enter the command exactly as it appears.

Format for Commands and Functions

This manual uses the following format conventions for documenting commands and functions.

NOTE

The command and function descriptions do not necessarily include each of the following sections.

NAME

The name of the command or function, along with a brief description of what it does.

SYNOPSIS

The syntax of the command or function. The following chart explains the notations used in the synopsis.

The Notations Used in the Synopsis

Notation	Meaning
< >	Angled brackets enclose terms in a command line that you must replace with literal values pertinent to your own service. In the following example, you must type in the following command line and replace the generic “service“ with the name of your service: **`mv slpi_<service> slpi_<service>.old`**

Notation	Meaning
[ ]	Square brackets enclose command argument choices that are optional. For example: **cflow [*-r*] [*-ix*] [*-i_*] [*-d num*] files**
|	The vertical bar separates mutually exclusive arguments from which you choose one. For example: **command [*arg1* | *arg2*]** You can use either `arg1` or `arg2` when you issue the command.
...	Ellipsis indicates that you can have many arguments on a single command line. For example, **command [*arg1, arg2, arg3,...*]**
$	In sample commands, the dollar sign is sometimes used for the shell command prompt. Your system prompt might differ. Although a prompt is sometimes shown at the beginning of a command line as it would appear on your screen, you do not type it.

NOTE

Dots, brackets, and braces are not literal characters; you should **not** type them. Any list or set of arguments can contain more than two elements. Brackets and braces are sometimes nested.

DESCRIPTION

A detailed description of the command or function. In command descriptions, this section also contains descriptions of the command's arguments. In function descriptions, this section also contains descriptions of the function's input or output parameters. An *input parameter* defines data that the application programmer must provide to the function (for example, the name of an application). An *output parameter* defines data that the function provides or returns (for example, a pointer to a particular data structure).

EXAMPLES

Examples of usage.

FILES
A list of the files that must be included in any program that uses this function.

NOTES
Hints about how to use the command or function.

RETURN VALUES
Values returned by the command or function.

SEE ALSO
A list of related information.

Related Manuals

Refer to the following Stratus manuals for related documentation:

- The *SINAP Products Glossary* (R8010) contains definitions for terms and acronyms used across the line of SINAP/SS7 products.
- The *SINAP/SS7 User's Guide* (R8051) provides instructions for configuring and managing a SINAP/SS7 installation.
- The *SINAP/SS7 ISDN User Part (ISUP) Guide* (R8053) provides instructions for configuring ISUP applications on the SINAP/SS7 system.
- The *SINAP/SS7 Technical Overview* (R8055) contains an overview of the SINAP/SS7 product.
- The *SINAP/SS7 Installation Guide* (R8060) provides instructions for installing the SINAP/SS7 software on a UNIX system.
- The *SINAP/SNMP MIB Guide* (R8065) for detailed information on SNMP application installation, configuration, and operation.
- The appropriate hardware and operating system manuals supplied with your configuration.

A Note on the Contents of Stratus Manuals

Stratus manuals document subroutines and commands of the user interface. Any other commands and subroutines contained in the operating system are intended solely for use by Stratus personnel and are subject to change without warning.

Accessing Documentation

SINAP product documentation is provided on CD-ROM. You can request a documentation CD-ROM in either of the following ways:

- Call the CAC (see “Commenting on the Documentation”).
- If your system is connected to the Remote Service Network (RSN), add a call using the Site Call System (SCS). See the scsac(1) man page for more information.

When requesting a documentation CD-ROM, please specify the product and platform documentation you desire, as there are several documentation CD-ROMs available.

Commenting on the Documentation

To provide corrections and suggestions for improving this documentation, send email to Comments@stratus.com. If it is possible, please include the title and part number from the Notice page and the page numbers.

This information will assist Stratus Publications in making any needed changes to the documentation. Your assistance is most appreciated.

Contacting the CAC

If you need assistance, contact your local systems engineer, or telephone the Stratus Customer Assistance Center (CAC) that services your area. If you cannot reach the center that services your area, contact the CAC in the United States.

The table below lists the CAC telephone numbers, all of which are available 24 x 7. For the most current list of CAC telephone numbers, see the following Web site: `http://www.stratus.com/support/cac`.

Worldwide CAC Telephone Numbers *(Page 1 of 2)*

Customer Assistance Center (CAC)	**Telephone Numbers**
North America, Central America, and South America	800-221-6588 (toll-free within USA or Canada)
	800-828-8513 (toll-free within USA or Canada)
	+1-978-461-7200 (Maynard, MA; for local and international direct)
	+1-602-852-3200 (Phoenix, AZ; for local and international direct)
Australia	1800-025-046 (toll-free within Australia)
Belgium*	+32 2-512-63-70 (Dutch language)
	+32 2-512-77-06 (French language)
France	+33 (0) 1-41-20-37-08
Germany	+49 (0) 6196-472518
Hong Kong	800-900-938 (toll-free within Hong Kong)
Italy	+39 02-467440-216
Japan	0120-725530

Worldwide CAC Telephone Numbers *(Page 2 of 2)*

Customer Assistance Center (CAC)	**Telephone Numbers**
Mexico	+52 55-5553-4792
The Netherlands*	+31 (0) 346-582-112
New Zealand	0800-443-051 (toll-free within New Zealand)
People's Republic of China	+86 139-010-39512 (Beijing)
	+86 21-63877700 (Shanghai)
Singapore	1800-2727482 (toll-free within Singapore)
South Africa	+27 11-2675-709
Spain	+34 91-383-4294
United Kingdom	+44 (0) 1784-246056

*For the countries of Belgium, Denmark, Luxembourg, The Netherlands, Norway, and Sweden, you can also use the following toll-free number to call after hours: 00800-000-99999. Your call will be directed to Phoenix Support Coordination.

NOTES

1. The plus sign (+) indicates that an international access code is required. The access code for international calls varies from country to country; in the United States, it is 011.
2. When you call from within the same country as the CAC office, be sure to include any necessary long distance or STD call prefix. If you use an international telephone number within the same country, you must replace the country code with the necessary prefix. For example, within the United States, callers dial 1-800-221-6588.
3. The telephone numbers in the preceding list are for CACs operated by Stratus. If you receive service from a distributor of Stratus products, contact your distributor for instructions about obtaining assistance.

Chapter 1
Introduction

The *SINAP/SS7 Programmer's Guide* (R8052) documents the application programming interface (API) of the Stratus Intelligent Network Applications Platform (SINAP)/MultiStack product. It is intended for programmers who develop applications to run on the SINAP/SS7 system. This guide describes the features of the SINAP/SS7 system and the associated programming considerations. It describes the process of developing Message Transfer Part (MTP), Signaling Connection Control Part (SCCP), Transaction Capabilities Application Part (TCAP), and Integrated Services Digital Network (ISDN) User Part (ISUP) applications for the SINAP/SS7 system. Detailed descriptions are provided for functions in the Common Application Services Layer (CASL) library, structures, include files, and primitives.

What is SINAP/SS7?

The SINAP/SS7 product is a network-services development and implementation platform. The SINAP/SS7 software provides the Signaling System Number 7 (SS7) communications protocol, along with tools to aid in the development, testing, and implementation of applications that provide network services in an SS7 network. SINAP/SS7 comes in two sizes. The *SINAP* (unistack) product supports a single SS7 node, while the *MultiStack* product provides the ability to run up to four SINAP nodes on a single Stratus system, where each node is a single instance of SINAP and is configured as a separate point code in the same network or in different networks.

The SINAP/SS7 system functions as an end point, such as a service control point (SCP), node within the Advanced Intelligent Network (AIN). The SINAP/SS7 system can also function as an adjunct processor (AP) or service node (SN). The SINAP/SS7 system is designed to support multiple enhanced service applications, such as 800 number translation, simultaneously while providing optimal performance for individual applications.

The SINAP/SS7 system implements the SS7 protocol to provide communications between applications and network elements, such as remote SCPs or signaling transfer points (STPs). Client applications residing on the SINAP/SS7 system initiate queries to the SS7 network and respond to queries from other network elements. Typically, these queries are in the form of TCAP messages. However, applications can also be configured to interface to the SS7 network at the MTP or the SCCP layers. In addition, the SINAP/SS7 system provides capabilities designed to simplify a customer's development, testing, implementation, and troubleshooting of new applications.

NOTES

1. Throughout this document, *the SINAP/SS7 system* refers to either the SINAP or MultiStack product, whichever is running on your system. SINAP *node* and SINAP *stack* refer to a single instance of the SINAP product running on your system. With the MultiStack product you can have multiple SINAP stacks on your system; with SINAP, you can have only one SINAP stack. SINAP *variant* or *network variant* refers to the type of SS7 protocol (ANSI, CCITT, TTC, NTT, or China) configured to run on a particular SINAP node.
2. Although the name of the International Telegraph and Telephone Consultative Committee (CCITT) was changed to the International Telecommunications Union (ITU) and its standards are now referred to as ITU-T recommendations, the SINAP and MultiStack products continue to refer to this protocol version as CCITT.
3. This document uses the generic term *UNIX* to refer to all supported varieties of UNIX, such as the HP-UX operating system. Differences between the varieties are noted in the text.

The *SINAP/SS7 Technical Overview* (R8055) contains a detailed description of the SINAP/SS7 product, its subsystems, and its features. It is highly recommended that you read the technical overview and have a firm understanding of the functionality before you use the SINAP/SS7 product and this manual.

Chapter 2
Application Programming Interface (API)

This chapter presents information about the SINAP/SS7 API. It contains the following sections.

- "API Overview" introduces the SINAP/SS7 application programming interface (API).
- "CASL Function Types" provides an overview of the types of functions in the Common Application Services Layer (CASL) library.
- "CASL Structure Types" describes the structures that the SINAP/SS7 system uses to store and pass information.
- "SINAP/SS7 Include Files" describes the include files containing SINAP/SS7 definitions.
- "SS7 Primitives" describes the types of primitives used by applications that run in an SS7 network.
- "SS7 Message Processing" provides background information about how the SINAP/SS7 system interacts with the network to process SS7 messages.
- "Interprocess Communications (IPC)" describes the SINAP/SS7 IPC mechanism, which provides a way for SINAP/SS7 client applications and SINAP/SS7 subsystems to communicate.

API Overview

The API provides access to the SINAP/SS7 platform, which in turn provides access to the SS7 suite of protocols running on the Stratus UNIX system. You use the API to develop applications that run in an SS7 network by writing C-language code that includes calls to appropriate CASL functions. These applications run on the SINAP/SS7 platform and use the services it provides. Such applications are considered clients of the SINAP/SS7 system, or *client applications*.

Through the CASL, the SINAP/SS7 platform provides access to the Message Transfer Part (MTP), Signaling Connection Control Part (SCCP), Transaction Capabilities Application Part (TCAP), and Integrated Services Digital Network User Part (ISUP) layers of the SS7 protocol. A client application uses the services of a particular SS7 protocol layer and is therefore considered a *user part* or *user* of that layer. For example, an application that interfaces with the SINAP/SS7 system at the MTP layer is considered an *MTP user* or *MTP user part*, just as an application that interfaces at the TCAP layer is considered a *TCAP user* or *TC user*.

NOTE

Throughout this manual, references to an application indicate the boundary at which the application interfaces with the SINAP/SS7 system. For example, the term *MTP application* refers to an application that interfaces with the SINAP/SS7 system at the MTP layer, *TCAP application* refers to an application that interfaces with the SINAP/SS7 system at the TCAP boundary, and so on.

Each SS7 protocol layer provides a particular set of services. In addition, each layer uses the services of the layers below it. Figure 2-1 shows how the layers of the SS7 protocol relate to one another. For example, an SCCP application uses the services of the SCCP layer, which in turn uses the services of the MTP layer. A TCAP application uses the services of the TCAP layer, which then uses the services of the SCCP layer, which in turn uses the services of the MTP layer. An ISUP application can use the services of the SCCP and MTP layers.

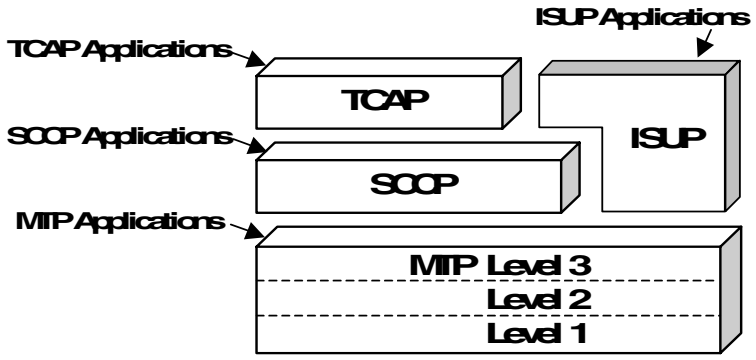

Figure 2-1. SS7 Protocol Layer Interaction

As a programmer, you need be concerned only with the API of the layer at which your application interfaces with the SINAP/SS7 system. If you are developing a TCAP application, you need concern yourself only with the TCAP API. The SINAP/SS7 system automatically performs the necessary processing for the TCAP to access the services of the lower layers (SCCP and MTP). For example, you would code a TCAP application to call the CASL function `ca_put_tc()` to send a TCAP component to a remote user that might or might not be implemented as a TCAP user. The SINAP/SS7 system then automatically packages the TCAP component in a message signaling unit (MSU) and calls the function `ca_put_msu()` to send the MSU to the SS7 network for delivery to the remote user. You need not code the application to call `ca_put_msu()`.

NOTE

Unless otherwise specified, header files can be found in the `$SINAP_HOME/Include` directory.

CASL Function Types

The CASL library provides the following types of functions, each of which is described in detail in Chapter 6, ‘‘CASL Function Calls.”

- SINAP/SS7 Management Functions
- SS7 Functions
- ISUP Services Functions
- Inter Process Communications (IPC) Functions
- Connection-oriented Services Functions
- Load Control Functions
- BITE Functions
- Miscellaneous Functions

SINAP/SS7 Management Functions

SINAP/SS7 management functions are used to perform various types of management functions for the client application.

Registration and Termination	`ca_register()`, `ca_terminate()`, `ca_withdraw()`
Command and Reply	`ca_put_cmd()`, `ca_put_reply()`
Event Reporting	`ca_put_event()`
Health Check	`ca_health_chk_req()`, `ca_health_chk_resp()`

SS7 Functions

SS7 functions are used to communicate with the SS7 network. MTP and SCCP applications use these functions to pass MSUs to and from the SS7 network. TCAP applications use SS7

functions to respond to subscriber or network management requirements by generating and transferring messages to the MTP, and processing queries from the SS7 network.

SS7 Message Handling (MTP and SCCP)	`ca_get_msu()`, `ca_put_msu()`, `ca_flush_msu()`, `ca_get_opc()`, `ca_get_msu_noxudt()`, `ca_lookup_gt()`
Component Handling	`ca_get_tc()`, `ca_get_tc_ref()`, `ca_put_tc()`, `ca_alloc_tc()`, `ca_dealloc_tc()`, `ca_process_tc()`, `ca_dist_cmd()`, `ca_cust_dist_cmd()`
Dialogue/Transaction Processing	`ca_get_dial_id()`, `ca_rel_dial_id()`, `ca_get_trans_id()`, `ca_rel_trans_id()`

ISUP Services Functions

ISUP services functions are used to develop applications that use ISUP services to send and receive ISUP messages. For detailed information on these functions, see the *SINAP/SS7 ISDN User Part (ISUP) Guide* (R8053).

IPC Functions

IPC functions are used for internal communication between SINAP/SS7 client applications and SINAP/SS7 subsystems.

IPC Key Processing	`ca_get_key()`, `ca_check_key()`, `ca_swap_keys()`, `ca_ascii_u32()`, `ca_u32_ascii()`
MML Command Handling	`ca_put_cmd()`
IPC Message Handling	`ca_get_msg()`, `ca_put_msg()`, `ca_put_reply()`
Deferred IPC Message Handling	`ca_put_msg_def()`, `ca_restart_timer()`, `ca_cancel_def()`

Connection-oriented Services Functions

Connection-oriented services functions are used in applications that use connection-oriented services to establish and maintain connections with other applications to exchange data.

Connection-Oriented Processing	`ca_get_sc(), ca_put_sc()`

Load Control Functions

Load Control functions are used for implementing the load control facility in an application. Load control helps maintain an application's throughput in spite of severe network congestion.

Load Control Management	`ca_setup_locon()`, `ca_inquire_locon()`
Load Control Processing	`ca_enable_locon()`, `ca_disable_locon()`, `ca_invoke_locon()`, `ca_exit_locon()`

BITE Functions

BITE functions are used for monitoring and debugging client applications.

Monitor	`ca_enable_mon()`, `ca_disable_mon()`
Intercept	`ca_enable_intc()`, `ca_disable_intc()`
Debug	`ca_dbg_dump()`, `ca_dbg_display()`

Miscellaneous Functions

The CASL library also contains other miscellaneous functions for character string conversions and alarms.

Character String Conversions	`ca_pack(), ca_unpack()`

CASL Structure Types

The SINAP/SS7 system uses the following types of data structures to manage interaction and data exchange between a client application and the CASL.

- The `I_Block` structure is used for IPC activities
- The `M_Block` structure is used to transport MSUs through the SS7 network
- The `T_Block` structure is used to transport TCAP components to the TCAP layer
- The connection-oriented structures are used for connection-oriented services
- The ISUP services structures are used to pass ISUP messages

For information on ISUP services structures, see the *SINAP/SS7 ISDN User Part (ISUP) Guide* (R8053).

The remainder of this section briefly describes each of these structures. Each structure is described more fully in Chapter 6 in the description of the CASL function with which the structure is used. Unless stated, header files are defined in the directory `$SINAP_HOME/Include`.

`I_Block` – IPC Unit of Exchange

The `I_Block` structure contains information used for IPC functions. An IPC message is composed of the following parts:

- A CASL control part
- A transaction part
- A timestamp part
- A node part
- An originator key part
- A destination key part
- A message body part

The `i_block_t` structure is defined in the include file `$SINAP_HOME/Include/iblock.h`.

You must allocate memory space for `I_Block` structures to be used in either the `ca_put_msg` or the `ca_get_msg` function. The SINAP/SS7 system does not provide the ability to allocate `i_block_t` structures—you must allocate them through a variable declaration or `malloc` call.

M_Block – SS7 MSU-Level Unit of Exchange

An m_block_t structure contains a single MSU and is used in the SINAP/SS7 system and throughout the SS7 I/O subsystem to communicate MTP- and SCCP-level protocol packets to client applications. The M_Block structure is defined in the mblock.h include file.

It consists of the following elements:

- A CASL control part
- A timestamp part
- A priority part
- A Built-In Test Environment (BITE) control part
- A TCAP control part
- An SCCP control part
- An MTP control part
- An MSU data part

You must allocate memory space for an M_Block structure before it can be used in the ca_put_msu function. In addition, since the SINAP/SS7 system dynamically allocates the M_block structure pointed to by the ca_get_msu function, the M_Block structure might not exist after the next call to ca_get_msu. If you require the MSU to exist after succeeding ca_get_msu calls, you must explicitly allocate space for it and copy it yourself.

T_Block – SS7 TCAP-Level Unit of Exchange

A T_Block structure is used by a TCAP application to exchange data with the CASL. It contains all information necessary to initiate and maintain communication with another TCAP user. The T_Block structure, which is defined in the include file $SINAP_HOME/Include/tblock.h, contains the following types of information.

- Component-handling information
- Dialogue-handling information (CCITT/TTC/NTT/China)
- Transaction-handling information (ANSI)
- Addressing information (for both the local and remote TCAP users)
- Data and control information
- Priority and sequence control
- Error and problem codes

Although TCAP applications do not use the M_Block data structure directly, the SINAP/SS7 system packages T_Block information into an MSU, which is sent to the SS7 network for delivery to the remote user.

NOTE

The `T_Block` structure is part of a dynamically allocated memory pool and must be allocated. The SINAP/SS7 system provides the `ca_alloc_tc` function that you must call to assign and use a `T_Block` structure from the allocated pool. The `tc_count` field of the `register_req_t` structure defines the number of `T_Block` structures to allocate for a TCAP application.

TCAP Application-Context Structures

An application that implements the 1993 TCAP standards uses the following structures to transmit information for an application-context dialogue:

- `tc_association_t`—Contains the data comprising the dialogue portion of the MSU. The dialogue portion defines the application-context name and optional user information to be used for the application-context dialogue. It also contains the `acn_t` and `tc_user_data_t` structures. It is defined in the `tblock.h` include file.
- `acn_t`—Contains the application-context name to be used for the application-context dialogue.
- `tc_user_data_t`—Contains optional user information for the application-context dialogue.

With the exception of a few fields initialized by TCAP, the fields in these structures are initialized by the *TC user*, which is the application initiating an application-context dialogue or sending an MSU that is part of a dialogue. When your application is the TC user, it must initialize the fields in the application-context structures. When your application is processing an incoming MSU that initiates or is part of an application-context dialogue, the other application is the TC user and, as such, will have initialized the fields in the application-context structures.

Connection-Oriented Structures

The following CASL structures are used for connection-oriented services:

- `sccp_ipc_t`—Passes interprocess communications (IPC) messages between the local application and the SCCP-SCOC process. This structure contains several structures, each of which passes a particular type of message. The `SCCP_ipc_t` and all structures within it are defined in the `SCOC_prims.h` include file.
- `sccp_prim_t`—An internal structure that conveys information about large messages, such as the message size and buffer location. It is defined in the `mblock.h` include file.
- `sccp_cldclg_t`—Contains information about the SCCP called or calling party address for a connection-oriented message. It is defined in the `mblock.h` include file.

- `sccp_dt1_t`—Transports a data-form-1 message. It is defined in the `sccphdrs.h` include file.
- `sccp_dt2_t`—Transports a data-form-2 message. It is defined in the `sccphdrs.h` include file.
- `sccp_expdata_t`—Transports a message containing expedited data. It is defined in the `sccphdrs.h` include file.

When sending an IPC message to the SCCP-SCOC process or a data MSU to another application, your application must initialize the appropriate structures. When your application is processing an incoming MSU or IPC message, the structures will have been initialized by the other application, the SCCP-SCOC process, or the SINAP/SS7 system.

When an ISUP services application processes an incoming ISUP message, the structure fields are initialized according to the ISUP message from the remote application. For information on ISUP Services, see the *SINAP/SS7 ISDN User Part (ISUP) Guide* (R8053).

SINAP/SS7 Include Files

The following table describes the SINAP/SS7 include files that are located in the directory `$SINAP_HOME/Include`. For information on ISUP Services include files, see the *SINAP/SS7 ISDN User Part (ISUP) Guide* (R8053).

Table 2-1. SINAP/SS7 Include Files *(Page 1 of 10)*

Include File	Description
`ansi_variant.h`	Contains a definition of the global variable `L_ANSI`, which is required for applications that run on the ANSI network variant of the SINAP/SS7 system. Contains no other include files.
`arch.h`	Defines the architectural characteristics of the Stratus UNIX system(s). A copy of this file must exist on each of the environments on which the SINAP/SS7 system runs. Contains no other include files.
`bidb.h`	The `#define SIO_LABEL_LEN_N8` was called to accommodate China network variant requirements.
`bitemon.h`	Contains definitions of the BITE monitor IDs used by the CASL for monitoring IPC messages. Contains no other include files.
`blkhdr.h`	Contains a definition for the title CASL control structure. Should be used only in `M_Block` and `I_Block` data structures. Contains no other include files.

Table 2-1. SINAP/SS7 Include Files *(Page 2 of 10)*

Include File	Description
`ca_error.h`	Contains a list of all CASL error messages. Contains the include file `<variant.h>`. For the China network variant, a `#define CA_ERR_GETSC_NG` indicates an invalid message type for `CA_GET_SC` (COF).
`ca_glob.h`	Contains pointers to all global tables located in shared memory. This includes the global variable `CHINA_APPL_VER="CHINA"` to implement the China network variable. For this file to be included, an application must also include the following files: • `<sys/time.h>` • `sinap.h` • `sysdefs.h` • `register.h` • `ipctbl.h` • `network.h` • `sccp.h` • `irt3.h` • `ort3.h` • `mtp.h` • `bitemon.h` • `measure.h` • `locon.h`. References the include files: • `<sys/types.h>` • `<locon.h>` • `<register.h>` • `<variant.h>`.
`cadbg.h`	Contains the definitions of the structures, global variables, and masks for displaying debug and trace information. Contains no other include files.

Table 2-1. SINAP/SS7 Include Files *(Page 3 of 10)*

Include File	Description
`casl.h`	This file contains the definition for `CASL_FLAG`. No other include file should contain this definition. This file contains the following definitions for the XUDT feature: • `CA_XUDT_HASH_SEED` - The XUDT reassembly hash seed, defined to be 128. • `ca_xudt_free` - The XUDT reassembly buffer free list structure contains pointers to the first and last entries in the list. • `ca_xudt_hash_table` - The hash table array. • `ca_xudt_reassemble` - The XUDT reassembly buffer structure that stores the message and pertinent information about the message being reassembled, such as the local reference number (LRN), originating point code (OPC), and originating subsystem number (SSN). References no other include files.

`caslinc.h`	The master CASL include file that contains these frequently used include files:
	`<sys/types.h>` `<sys/time.h>`
	`<errno.h>` `<sys/stream.h>`
	`<sys/stropts.h>` `<signal.h>`
	`<malloc.h>` `<unistd.h>`
	`<string.h>` `<stdlib.h>`
	`<stdio.h>` `<arch.h>`
	`<sysdefs.h>` `<sinap.h>`
	`<blkhdr.h>` `<timestamp.h>`
	`<iblock.h>` `<register.h>`
	`<ipctbl.h>` `<ca_error.h>`
	`<network.h>` `<event3.h>`
	`<sccp.h>` `<irt3.h>`
	`<ort3.h>` `<mtp.h>`
	`<mblock.h>` `<bitemon.h>`
	`<measure.h>` `<terminate.h>`
	`<s7signal.h>` `<event.h>`
	`<tblock.h>` `<treatment.h>`
	`<sysshm.h>` `<command.h>`
	`<sinapintf.h>` `<locon.h>`
	`<cadbg.h>` `<ca_glob.h>`
	`<scmg-prims.h>`.

`ccitt_variant.h`	Contains a definition of the global variable `L_CCITT`, which is required for applications that run on the CCITT network variant of the SINAP/SS7 system. Contains no other include files.

Table 2-1. SINAP/SS7 Include Files *(Page 4 of 10)*

Include File	Description
china_variant.h	Contains definition of the global variable, L_China, which is required for applications that run on the China network variant of the SINAP/SS7 system. Contains no other include files.
client.h	Contains definitions of node management's client-management structures: • process_term_t • health_check_t • register_resp_t • terminate_resp_t Contains no other include files.
command.h	Contains definitions of the message types relevant to the command management interface of the SINAP/SS7 Node Management (for example, I_MTP_CHANGE, I_MTP_CHANGE_ACK, and so on). Contains no other include files.
cust_dist.h	Defines the ID values used to identify custom application distribution (CAD) functions and structure definitions for custom registration request and inquiries. Contains no other include files.
dl_chan_user.h	Contains definitions and structures used for allocating and deallocating channels for ISDN usage. Contains no other include files.
dr_incl.h	Contains definitions used by the SS7 device driver. Contains the include file <register.h>.
dr_minor.h	This file, which is used by the UNIX SS7 device driver, contains minor device number definitions. Contains no other include files.
eqpi_appl.h	Contains the definitions of EQPI (extended QPI provider interface) data structures and macro definitions required by ISDN-BRI applications. Contains no other include files.
event.h	Contains the data structure for the EVENT message, which contains information about a situation being reported to the SINAP/SS7 trouble management by a registered application process. Contains no other include files.
event3.h	Defines the structure of the data section of M_EVENT M_Block messages from UCOMM Level 2 and Level 3 management to MTP management. Contains no other include files.

Table 2-1. SINAP/SS7 Include Files *(Page 5 of 10)*

Include File	Description
`fts_dev.h`	HP-UX operating system source file not delivered with the normal release containing definitions and device drivers required to utilize FTS. (FTS determines I/O board level events such as breaking or restoration to service.)
`fts_info.h`	HP-UX operating system source file not delivered with the normal release containing definitions required to utilize FTS. (FTS determines I/O board level events such as breaking or restoration to service.)
`iblock.h`	Contains `I_Block` definitions and structure formats. (The `I_Block` is used to transport IPC messages.) For this file to be included, an application must also include the following files: • `sysdefs.h` • `sinap.h` • `timestamp.h` • `<sys/time.h>`. Contains the include file: `<blkhdr.h>`.
`ipctbl.h`	This file, the IPC process table, contains information about each SINAP/SS7 application process. As an application process registers with the SINAP/SS7 system, its registration parameters are written to the IPC process table. Contains no other include files.
`irt3.h`	(Inbound routing tables for MTP Level-3) The table is used by driver-resident MTP functions to find the link set and link number for the message coming from the input/output (I/O) adapter. Contains no other include files.
`locon.h`	Contains load control definitions (user and internal). Contains no other include files.

Table 2-1. SINAP/SS7 Include Files *(Page 6 of 10)*

Include File	Description
`mblock.h`	Contains definitions of `M_Block` message types and structure format. (The `M_Block` is used to pass MSUs between the driver and the user subsystems.) For this file to be included, an application must also include the following files: • `sysdefs.h` • `sinap.h` • `iblock.h` • `timestamp.h` • `<sys/time.h>`. Contains the include file `<variant.h>`. Contains the TTC variant's version of some generic structures, including: • `ttc_msu_t` • `ttc_sccp_user_t` • `ttc_snm_user_t` For the China network variant, a `#ifdef` sets `MAX_MTP_THRESHOLD` to `CCITT_MAX_MTP_THRESHOLD` instead of to the `ANSI_MAX_MTP_THRESHOLD`.
`measure.h`	Contains definitions of measurement-related data structures. Contains no other include files.
`measure3.h`	Defines the structure of the data section of the `MTP_MEASUREMENT_RESPONSE I_BLOCK` message from MTP management to node management and `M_MEASUREMENT_REQUEST M_Block` messages from the I/O adapter's Level 2 and Level 3 management to MTP management. Contains no other include files.
`mml.h`	Contains MML-related definitions and structures. Contains no other include files.
`mtp.h`	Contains segment definitions for MTP routing tables. Contains no other include files.
`mtpevents.h`	Defines the format and structure for all categories of MTP Level 3 events. Contains no other include files.
`mtptypes.h`	Contains definitions used by the MTP code for communication between I/O adapters and MTP management. Includes definition for constants and data structures used in original MTP code in terms of SINAP/SS7 definitions. Contains no other include files.

Table 2-1. SINAP/SS7 Include Files *(Page 7 of 10)*

Include File	Description
`network.h`	Network tables contain values for the configuration data. Node Management initializes and updates the tables. Contains the include file `<variant.h>`. For the China network variant, a `#ifdef` sets `MAX_MTP_THRESHOLD` to `CCITT_MAX_MTP_THRESHOLD` instead of to the `ANSI_MAX_MTP_THRESHOLD`.
`nmcmdata.h`	Contains definitions for the private data structures of the Node Management command management (nmcm) process. Contains no other include files.
`nmcmglob.h`	Contains definitions for the global variables used for the nmcm process. Include this file **after** the `<nmcmdata.h>` file. Contains no other include files.
`ort3.h`	Contains the outbound routing tables for MTP Level 3. Contains no other include files.
`prims3.h`	Defines the structure of MTP indication primitives (`i_block_t` messages). Contains no other include files.
`proc_tc.h`	Contains the definition for the structure used by the CASL `ca_process_tc()` function. This structure contains the address of each function called for each state and event. Contains no other include files.
`register.h`	Defines SINAP/SS7 application-registration data. Contains the include file `cust_dist.h`.
`s7signal.h`	Contains definitions of the SINAP/SS7 signals used for Stratus UNIX systems. Contains the include file `<signal.h>`.
`sccp-intrn.h`	Internal SCCP header file. Include this file **after** the `<scmg-prims.h>` file. Contains the include file `<variant.h>`.
`sccp.h`	Contains definitions and structure formats for the SCCP shared-memory segment. Contains no other include files.
`sccphdrs.h`	Contains the general SCCP header and the headers for SCCP connection requests. Contains no other include files.

Table 2-1. SINAP/SS7 Include Files *(Page 8 of 10)*

Include File	Description
`scmg-prims.h`	Contains definitions and structure formats for the SCCP management IPC primitives: `N-STATE`, `N-PCSTATE`, and `N-COORD`. For the China network variant, `#ifdef L_China` specifies the use of `CCITT_SCMG_PC_MAX` instead of `ANSI_SCMG_PC_MAX`. Contains the structure, `sccp_xudt_err_t`, which includes a definition for an `T_SCCP_XUDT_ERR_IPC` message. Include this file **after** the `command.h` file. Contains the include file `<variant.h>`.
`sinap.h`	Contains all SINAP/SS7 system application and process definitions. Contains the include file `<variant.h>`. For the China network variant, a `#ifdef L_China` sets the `MAX_ROUTE_PER_RS` to `CCITT_MAX_ROUTE_PER_RS` to use the CCITT variant version.
`sinap_variant.h`	Contains a reference to the appropriate network variant include file: • `ccitt_variant.h` • `ttc_variant.h` • `ntt_variant.h` • `china_variant.h` • `ansi_variant.h` References the variant of the SINAP/SS7 system you are running.
`sinapintf.h`	Contains definitions of all global variables used by the SINAP/SS7 system and its client applications. This file contains conditional code such as `CASL_FLAG`, which is defined internally in the `casl.c` file. For this file to be included, an application must also include the following files: • `sinap.h` • `sysdefs.h` • `register.h`. Contains no other include files.

Table 2-1. SINAP/SS7 Include Files *(Page 9 of 10)*

Include File	Description
`sysdefs.h`	Contains the data-type definitions for the type of hardware on which SINAP/SS7 is running. This file contains the definitions for portable software. Architecture characteristics must be defined externally by means of the `arch.h` include file. This header file should be included in an application first. Contains the include file `"arch.h"` or `<arch.h>`.
`sysshm.h`	Contains definitions for system-shared memory variables for all processes. Contains the system option definitions for the MTP restart, time-controlled changeover (TCCO), and time-controlled diversion (TCD) features. Contains no other include files.
`tblock.h`	Contains the definition of the `T_Block` structure, which is used to pass data between a TCAP application and the CASL. Contains the include file `<variant.h>`.
`tcap.h`	Contains definitions for TCAP tables and the data structures used by the Component Sublayer (CSL) and the Transaction Sublayer (TSL). Contains the include file `<variant.h>`.
`tccom.h`	Contains definitions for XUDT messages.
`tcglob.h`	Contains definitions for TCAP globals and data structures. This file also declares the global pointer for various queues and arrays. Contains no other include files.
`terminate.h`	Contains definitions for the structure elements used by the CASL function `ca_terminate()`. This structure is copied into the `I_Block` data header. Contains no other include files.
`timestamp.h`	Contains definitions and structure formats for the timestamp included in `I_Block` and `M_Block` structures. Contains no other include files.
`treatment.h`	Contains the definitions of labels used for specifying values in the SINAP/SS7 trouble treatment table. Defines the `treatment_t` and `treat_t` data structures, which are used by trouble management to determine event treatment. Contains no other include files.
`ttc_variant.h`	Contains a definition of the global variable, `L_TTC`, which is required for applications that run on the TTC network variant of the SINAP/SS7 system. Contains no other include files.

Table 2-1. SINAP/SS7 Include Files *(Page 10 of 10)*

Include File	Description
`variant.h`	Contains all SINAP/SS7 variant definitions: • `V_CCITT` • `V_ANSI` • `V_TTC` • `V_NTT` • `V_China` Also contains the macro, `IS_China`. Contains the include file `"sinap_variant.h"`

SS7 Primitives

This section describes the various primitives used by the SINAP/SS7 system in IPC messages to communicate with the SS7 network. It contains sections on the primitives used by MTP, SCCP, and TCAP. For information on how these and other primitives are used for error handling, see ‘‘Error Handling’’ in Chapter 3.

MTP Primitives

Table 2-2 lists the MTP primitives that inform the SS7 user part about the accessibility of the remote signaling points. These primitives are passed in IPC messages and are defined in the `$SINAP_HOME/Include/iblock.h` file. The structures associated with the primitives are defined in the include file `$SINAP_HOME/Include/prim3.h`.

Table 2-2. MTP Primitives

Primitive	Description
`I_MTP_PAUSE`	Informs the user part that the remote signaling point is not accessible and that the user part should stop traffic towards the destination.
`I_MTP_RESUME`	Informs the user part that the remote signaling point became accessible and that the user part can start traffic towards the destination.
`I_MTP_STATUS`	This primitive informs the user part about the congestion status of the remote signaling point. The user part handles the congestion information. This primitive contains several unavailability-cause parameter values for user part unavailable (UPU) messages that indicate why a UPU message was generated.

SCCP Primitives

Table 2-3 describes the SCCP primitives, which are defined in an `mblock.h` or `iblock.h` include file in the `$SINAP_HOME/Include` directory. The structures and values associated with the IPC primitives are defined in the file `$SINAP_HOME/Include/scmg-prims.h`.

Table 2-3. SCCP Primitives *(Page 1 of 2)*

Primitive	Description
`I_N_STATE_REQ`	This primitive is passed in an `i_block_t` structure and is sent by the user to instruct the SINAP/SS7 system to send state information regarding availability to the remote stack. The following state values are passed: • `SCMG_UIS` - This is a user-in-service (UIS) message that an application issues each time it comes online. • `SCMG_UOS` - This is a user out-of-service (UOS) message that an application issues when it goes offline. These messages contain unavailability cause parameter values indicating why the UPU message was generated.
`I_N_STATE_INDIC`	This primitive is passed in an `i_block_t` structure and is sent by the user to the SINAP stack when the SINAP/SS7 system receives state information from the remote stack. The state values are passed in the following messages: • `SCMG_UIS` - This is a user-in-service (UIS) message that an application issues each time it comes online. • `SCMG_UOS` - This is a user out of service (OUS) message that an application issues when it goes offline. These messages contain unavailability cause parameter values indicating why the UPU message was generated.
`I_N_PCSTATE_INDIC`	This primitive is passed in an `i_block_t` structure and indicates whether the signaling point is accessible, inaccessible, or congested. The primitive is generated when SCCP management receives an `MTP_PAUSE`, `MTP_RESUME`, or `MTP_STATUS` message from MTP management. Point code state values generated are: • `SCMG_ACCESSIBLE` • `SCMG_INACCESSIBLE` • `SCMG_CONGESTED`

Table 2-3. SCCP Primitives *(Page 2 of 2)*

Primitive	Description
`I_N_COORD_<TYPE>`	This primitive is passed in an `i_block_t` structure. A replicated subsystem uses a type of `I_N_COORD` primitive whenever it wants to withdraw from the network. The primitive exists in one of the following four forms: • `I_N_COORD_REQ` - As a *request* when the originating user requests permission to go out of service • `I_N_COORD_INDIC` - As an *indication* when the request to go out of service is delivered to the originator's replicate • `I_N_COORD_RESP` - As a *response* when the originator's replicate indicates it has sufficient resources to let the originator go out of service • `I_N_COORD_CONF` - As a *confirmation* when the originator is informed it can go out of service.
`SC_N_UNITDATA`	This primitive is passed in an `m_block_t` structure. The TCAP and SCCP exchange data by means of the `SC_N_UNITDATA` primitive, which is invisible to the TCAP application. • For incoming messages, the TCAP decodes SCCP `SC_N_UNITDATA` indications and maps them to TCAP primitives, which are sent to TCAP dialogue- or transaction-handling procedures. • For outgoing messages, the TCAP decodes TCAP primitives and maps them to SCCP `SC_N_UNITDATA` indications which are sent to the SCCP.

Connection-Control Primitives

The following primitives are related to connection-oriented control. The primitives are divided into two types: those used in IPC messages and those used in data MSUs. For more information, see ''Connection-Oriented Control Primitives Used in IPC Messages'' in Chapter 3.

Primitives Used in IPC Messages

The following connection-oriented control primitives are used in IPC messages passed between the local application and the SCCP-SCOC process. These primitives define the IPC message type.

Table 2-4 describes the connection-control primitives that you can include in the IPC messages that the local application sends to SCCP-SCOC, and the `sccp_ipc_t` structure in which the IPC message is defined. When sending one of these IPC messages to SCCP-SCOC, your application must set the IPC message's `sccp_ipc_t.i_block_t.ipc_trans_t.msg_type` field to one of the values listed in

the column labeled “Primitive.” In addition, your application must initialize the `sccp_ipc_t` structure listed in the corresponding column labeled “Structure.”

Table 2-4. Outgoing Connection-Control Primitives

Primitive	Structure	Purpose
`I_N_CONNECT_REQ`	`scoc_con_req_t`	Request to establish a connection with a remote application
`I_N_CONNECT_RES`	`scoc_con_res_t`	Response to accept a remote application’s request to establish a connection
`I_N_RESET_REQ`	`scoc_res_req_t`	Request to initiate a reset procedure to reinitialize sequence numbers for the connection (class-3 services only)
`I_N_RESET_RES`	`scoc_res_res_t`	Response to accept a remote application’s request to initiate a reset procedure to reinitialize sequence numbers for the connection (class-3 services only)
`I_N_DISCONNECT_REQ`	`scoc_dis_req_t`	Request to disconnect (release) the connection
`I_SCOC_GET_CONNID`	`scoc_get_connid_t`	Request to obtain a connection ID for a connection

Table 2-5 describes the connection-control primitives that can be included in the IPC messages that the local application receives from SCCP-SCOC, along with the `sccp_ipc_t` structure in which the IPC message is defined. For incoming IPC messages, your application should examine the value of the `i_block_t.ipc_trans_t.msg_type` field to determine whether the message is a connection-oriented message. If the field’s value matches one of the values in the column labeled “Primitive,” the message is a connection-oriented message. The application should then read the message by examining the `sccp_ipc_t` structure listed in the corresponding column labeled “Structure.”

Table 2-5. Incoming Connection-Control Primitives

Primitive	Structure	Purpose
`I_N_CONNECT_CON`	`scoc_con_con_t`	Remote response accepting a local application’s connection request

Table 2-5. Incoming Connection-Control Primitives

Primitive	Structure	Purpose
I_N_CONNECT_IND	scoc_con_ind_t	Remote request to establish a connection
I_N_RESET_CON	scoc_res_con_t	Remote response to accept a local application's request to initiate a reset procedure to reinitialize sequence numbers for the connection (class-3 services only)
I_N_RESET_IND	scoc_res_ind_t	Remote request to initiate a reset procedure to reinitialize sequence numbers for the connection (class-3 services only)
I_N_DISCONNECT_IND	scoc_dis_ind_t	Remote request to disconnect (release) the connection; can be a response to a connection request
I_SCOC_CID_RESULT	scoc_cid_result_t	SCCP-SCOC response to request for connection ID

Data Primitives Used in Data MSUs

The connection-oriented control data primitives are used in the data MSUs passed between local and remote applications. The primitives define the type of data in the MSU. For more information about any of these primitives, see the ITU-T Recommendations Q.712.

Include the connection-oriented data primitives, listed in Table 2-6, in the data MSUs the local application sends to the remote application. Your application must set the `mblock_t.ud_ccitt_msu_t.mtp_ud.ccitt_sccp_user_t.msg_type` field to one of the values in the column labeled "Primitive" to define the MSU's data type. In addition, your application must initialize the `sccp_ipc_t` structure listed in the corresponding column labeled "Structure."

Table 2-6. Outgoing Connection-Oriented Data Primitives

Primitive	Structure	Purpose
SC_DATA_FORM1	sccp_dt1_t	Sends a data-form-1 message
SC_DATA_FORM2	sccp_dt2_t	Sends a data-form-2 message

Table 2-6. Outgoing Connection-Oriented Data Primitives

Primitive	Structure	Purpose
SC_EXPEDITED_DATA	sccp_expdata_t	Sends expedited data, which is a data-form-2 message that bypasses the flow-control settings defined for the connection

Table 2-7 describes the connection-oriented data primitives that can be included in the data MSUs received by the local application. Your application should examine the value of the `m_block_t.ud.ccitt_msu_t.mtp_ud.ccitt_sccp_user_t.msg_type` field. If the field's value matches one of the values in the column labeled "Primitive," the MSU contains data for a connection-oriented message. The application should then read the data by examining the `sccp_ipc_t` structure listed in the corresponding column labeled "Structure."

Table 2-7. Incoming Connection-Oriented Data Primitives

Primitive	**Structure**	**Purpose**
SC_DATA_FORM1	sccp_dt1_t	Contains a data-form-1 message
SC_DATA_FORM2	sccp_dt2_t	Contains a data-form-2 message
SC_EXPEDITED_DATA	sccp_expdata_t	Contains expedited data, which is a data-form-2 message that bypasses the flow-control settings defined for the connection
SC_RESET_REQUEST	sccp_resetreq_t	Remote request to initiate a reset procedure to re-initialize sequence numbers
SC_RELEASED	sccp_rlsd_t	Remote request to release the connection and associated resources

TCAP Primitives

TCAP primitives are divided into the following two classes:

- Transaction Capabilities (TC) primitives are related to transaction handling. Their purpose is to request or indicate facilities of the layer or sublayer underlying them in relation to message transmission or transaction handling.

NOTE

The CCITT, TTC, NTT, and China variants of the SINAP/SS7 system use dialogue-handling primitives instead of transaction-handling primitives. Their functions are basically the same.

- Component-handling primitives handle operations and replies; they do not require facilities from the underlying layer or sublayer.

In addition to these types of primitives, the TCAP and SCCP use the SC_N_UNITDATA primitive to exchange data. See the preceding section "SCCP Primitives," for more information about this primitive.

Dialogue-Handling Primitives (CCITT/TTC/NTT/China)

The transaction capabilities (TC) primitives, shown in Table 2-8, are related to dialogue handling. Their purpose is to request or indicate facilities of the layer or sublayer underlying them in relation to message transmission or dialogue handling.

Table 2-8. Transaction Capabilities Primitives (CCITT/TTC/NTT/China)

Primitive	Description
TC_UNI	Requests or indicates an unstructured dialogue.
TC_BEGIN	Begins a dialogue, indicating the beginning of a multicomponent transaction between two TC users.
TC_CONTINUE	Continues a dialogue and indicates that the TCAP components are packaged in the MSU and sent on the SS7 network.
TC_END	Ends a dialogue and indicates that the collected TCAP components are packaged in the MSU and sent on the SS7 network.
TC_U_ABORT	Allows a TC user to terminate a dialogue abruptly, without transmitting any components. All related information is discarded.
TC_P_ABORT	Informs the TC user that one of the protocol sublayers upon which the receiving TC user resides will terminate a specified dialogue.
TC_NOTICE	Informs the TC user if the service requested cannot be provided.
TC_REQUESTX	This primitive ensures components are carried in an XUDT message if the application is registered at the SS7 input boundary TCAPX.

Transaction-Handling Primitives (ANSI)

Table 2-9 describes the transaction-handling primitives for the ANSI network variant of the SINAP/SS7 system.

Table 2-9. Transaction-Handling Primitives (ANSI) *(Page 1 of 2)*

Primitive	Description
TC_UNI	Sends information to another TCAP-based client application; no reply is expected.
TC_QRY_W_PERM (query with permission)	Initiates a TCAP transaction between two TCAP-based client applications. The use of this primitive enables the destination node to end the TCAP transaction. When a single TCAP message consists of multiple TCAP components, the use of this primitive tells the TSL to assemble the message. The TSL assembles the message by concatenating all of the TCAP components into a single MSU, which is then delivered to SCCP.
TC_QRY_WO_PERM (query without permission)	Initiates a TCAP transaction between two TCAP-based client applications. The use of this primitive prohibits the destination node from ending the TCAP transaction. When a single TCAP message consists of multiple TCAP components, the use of this primitive tells the TSL to assemble the message. The TSL assembles the message by concatenating all of the TCAP components into a single MSU, which is then delivered to SCCP.
TC_CONV_W_PERM (conversation with permission)	Continues a TCAP transaction. The use of this primitive enables the destination node to end the transaction. When a single TCAP message consists of multiple TCAP components, the use of this primitive tells the TSL to assemble the message. The TSL assembles the message by concatenating all of the TCAP components into a single MSU, which is then delivered to SCCP.
TC_CONV_WO_PERM (conversation without permission)	Continues a TCAP transaction. The use of this primitive prohibits the destination node from ending the transaction. When a single TCAP message consists of multiple TCAP components, the use of this primitive tells the TSL to assemble the message. The TSL assembles the message by concatenating all of the TCAP components into a single MSU, which is then delivered to SCCP.
TC_RESPONSE	Ends a TCAP transaction after transmitting any pending TCAP components. This primitive also releases the transaction ID (if it belongs to the TCAP).

Table 2-9. Transaction-Handling Primitives (ANSI) *(Page 2 of 2)*

Primitive	Description
TC_NO_RESPONSE	Causes the local TCAP-based client application to terminate the transaction **without** transmitting pending TCAP components, and release the transaction ID if it belongs to the TCAP. (The use of this primitive is agreed upon by both TCAP-based client applications before the transition is started.)
TC_U_ABORT	Allows a TCAP-based client application to terminate a transaction abruptly, without transmitting any components. All related information is discarded.
TC_P_ABORT	Informs the TCAP-based client application that the transaction is being terminated by one of the protocol sublayers upon which the receiving client application resides.
TC_NOTICE	Informs the TCAP-based client application if the requested service cannot be provided.

Component-Handling Primitives

Table 2-10 describes the component handling primitives for all network variants of the SINAP/SS7 system. The primitives that apply to only one or more variants are designated with the network variant to which they apply.

Table 2-10. Component Handling Primitives *(Page 1 of 2)*

Primitive	Description
TC_INVOKE (CCITT/TTCNTT//China)	This primitive enables one TCAP user to request services from another TCAP user. In the ANSI variant of the SINAP/SS7 system, the primitives TC_INVOKE_L and TC_INVOKE_NL are used in place of TC_INVOKE.
TC_INVOKE_L (invoke last) (ANSI)	This primitive lets one TCAP-based client application request that another perform an operation. The TCAP-based client application must specify information that the component handler needs to process the operation (such as TC user ID, invoke ID, correlation ID, class of operation, transaction ID, and a timer value). If no correlation ID is specified (always coded as Last), it indicates that there are no further responding components.
TC_INVOKE_NL (invoke not last) (ANSI)	This primitive is similar to TC_INVOKE_L except that further responding components are expected.

Table 2-10. Component Handling Primitives *(Page 2 of 2)*

Primitive	Description
TC_RESULT_L	This primitive returns only the result, or last part of the segmented result, of a successfully executed operation. This primitive provides the general mechanism for responding to TC_INVOKE (CCITT/TTC/NTT/China) or TC_INVOKE_L (ANSI).
TC_RESULT_NL	This primitive returns the nonfinal (not last) part of the segmented result of a successfully executed operation. This primitive is useful for providing segmented responses when the result is too large for a single component. It also provides the general mechanism for responding to TC_INVOKE (CCITT/TTC/NTT/China) or TC_INVOKE_NL (ANSI).
TC_U_ERROR	This primitive is issued by a destination TCAP-based client application to indicate that, although the message is valid, it cannot execute the requested operation (for example, because the SINAP/SS7 node database is unavailable). This primitive contains the component that could not be executed, along with an error code.
TC_U_REJECT (user reject)	This primitive indicates that the TCAP-based client application rejected a component because it was improperly formed. This primitive contains a problem code that indicates the reason the component was rejected.
TC_L_REJECT (local reject)	This primitive is issued by the local CSL to indicate that it received a TC_INVOKE (CCITT/TTC/NTT/China), TC_INVOKE_L (ANSI), or TC_INVOKE_NL (ANSI) primitive whose content or structure was invalid or illegal.
TC_R_REJECT (remote reject)	This primitive is issued by the remote CSL to indicate that it received a TC_INVOKE (CCITT/TTC/NTT/China), TC_INVOKE_L (ANSI), or TC_INVOKE_NL (ANSI) primitive whose content or structure was invalid or illegal.
TC_L_CANCEL (local cancel)	This primitive is issued by the CSL to inform the TCAP-based client application that a timer associated with a requested operation (component) has timed out.
TC_U_CANCEL (user cancel)	This primitive is issued by a TCAP-based client application to indicate that it wishes to cancel previously-requested operations (components).

ISUP Services Primitives

For information related to ISUP Services primitives, see the *SINAP/SS7 ISDN User Part (ISUP) Guide* (R8053).

SS7 Message Processing

This section provides background information about how SS7 messages are processed. It describes how the SINAP/SS7 system interacts with the SS7 network to process incoming and outgoing SS7 messages. It also provides information about how a SINAP/SS7 client application reads from the queue and describes a potential timing problem that arises when the SINAP/SS7 system is used in a multiprocessor environment, such as the one provided by UNIX.

SINAP/SS7 Interaction with the SS7 Network

This section provides background information describing the interaction between the SINAP/SS7 system and the SS7 network. As you develop SINAP/SS7 applications, you should consider the following:

- The SINAP/SS7 system uses input and output batch buffers, respectively, as holding queues for inbound and outbound `M_Blocks`. Inbound MSUs from the SS7 network are held on the input batch buffer until the application reads them. The maximum number of `M_Blocks` that the input queue can hold for an application is defined by the `max_msu_input_que` field of the `register_req_t` structure. If the application does not issue a read before the queue becomes full, the SINAP/SS7 system discards any additional `M_Blocks` that arrive. (An application reads from the queue by issuing a call to the `ca_get_msu()` function (MTP or SCCP) or the `ca_get_tc()` function (TCAP).)

  If an application calls `ca_get_msu()` or `ca_get_tc()` and there are no incoming MSUs on the input queue, the SINAP/SS7 system retrieves a batch of incoming MSUs from the SS7 SVR4 Streams driver. (A batch is equal to the number of `M_Blocks` defined by the `register_req_t` structure's `batch_count` field.)

- Outbound MSUs are stored in an output batch buffer until the buffer becomes full or until the application calls the `ca_flush_msu()` function. Then, the SINAP/SS7 system sends all pending outbound MSUs to the SS7 SVR4 Streams driver for transmission to the SS7 network. The maximum number of `M_Blocks` that the output queue can hold for an application is defined by the `register_req_t` structure's `max_msu_output_que` field.

Issuing Calls to Read from the Queue

When an application wants to read from the queue, it issues a call to the appropriate CASL function. For example, an MTP application issues a call to the `ca_get_msu()` function and a TCAP application issues a call to the `ca_get_tc()` function.

The `ca_get_msu()` and `ca_get_tc()` functions can be called in blocking or nonblocking mode. When called in *blocking mode*, normal application processing is suspended until the function actually reads from the queue. (If there is nothing on the queue, the function call must wait for something to arrive.) When called in *nonblocking mode*, the function returns an error if

there is nothing on the queue; normal application processing is not suspended as it is when the function is called in blocking mode.

The function's `fwait` parameter indicates whether to execute the function call in blocking or nonblocking mode. An `fwait` value of `1` causes the function call to execute blocking mode; an `fwait` value of `0` causes the function call to execute in nonblocking mode. The `fwait` parameter is passed by value, which means the calling process makes a copy of the parameter's value at the time of the call. Changing the parameter's value at a later time does not affect the original copy of the parameter value and thus cannot affect the behavior of the called function.

Blocking-Mode Timing Problem

In the UNIX multiprocessor environment, there is a certain amount of time (called a *timing window*) between when the `ca_get_tc()` function is called and when it actually reads the queue. When `ca_get_tc()` is called in blocking mode, a potential timing problem arises when a signal is generated during this timing window. This is because the application process suspends execution of the blocking-mode read in order to execute an interrupt-handler function to process the signal. After processing the signal, the application process resumes execution of the blocking mode read. However, because the `fwait` parameter is passed by value and not by reference, there is no way for the interrupt-handler function (or another application process) to change the value of `fwait` from blocking to nonblocking mode before resuming the blocking-mode read. Consequently, the application process is stuck in the blocking-mode read until something actually arrives on the queue.

Implementation of the `ca_get_tc_ref()` Function

The CASL function `ca_get_tc_ref()` provides a workaround to this blocking-mode timing problem. The `ca_get_tc_ref()` function is almost identical to the `ca_get_tc()` function; however, in place of the `fwait` parameter (which is passed by value), `ca_get_tc_ref()` uses the parameter `*prefwait`.

The `*prefwait` parameter points to the global variable `REFWAIT`, whose value is a Boolean indicator that specifies whether the function call is to execute in blocking or nonblocking mode: `1` specifies blocking mode and `0` specifies nonblocking mode. (`REFWAIT` is defined in the include file `sinapintf.h`.) Since `*prefwait` is only a pointer to the global variable `REFWAIT`, the interrupt-handler function (or another application process) can dynamically change the value of `REFWAIT` from blocking to nonblocking mode. For more information about the `ca_get_tc_ref()` function, see its description in Chapter 6.

Interprocess Communications (IPC)

The CASL provides interprocess communications by using UNIX message queuing. Interprocess communications can occur between two SINAP/SS7 client applications, between a client application and a SINAP/SS7 subsystem, or between two SINAP/SS7 subsystems. There are two basic functions for IPC: `ca_get_msg()` and `ca_put_msg()`. Both functions assume that the originator and destination are registered with SINAP/SS7 Node Management and that dedicated IPC queues are assigned to them.

When a client application registers with the SINAP/SS7 system, an IPC message queue is created. When the application is terminated, this queue is deallocated. Messages sent to a client application by means of the IPC facility are placed on this queue, where they remain until the application retrieves them by calling the CASL function `ca_get_msg()`. The CASL functions use the message queue with subsystems like Node Management for transparent IPC communication. The client application can use the message queue for its own IPC purposes.

Applications can send messages using the CASL function `ca_put_msg()`. The CASL then determines whether the application is authorized to send a message to the specified destination. If so, the message is copied to the destination queue using UNIX facilities. If the message cannot be sent, the CASL returns an error.

An application process can send a message that is delayed a specified amount of time before it is delivered to the destination IPC message queue. This mechanism is useful for time-out handling, for scheduling functions over a given time period (every *n* seconds or minutes), or for providing a delay between message sending and delivery. To initiate deferred message sending, use the `ca_put_msg_def()` function. To cancel one or more messages awaiting timer expiration, use the `ca_cancel_def()` function. To let the client application reset a timer (or set the timer to a new value) for all messages with a particular timer identifier, use the `ca_restart_timer()` function.

All IPC messages are stored in an `I_Block` structure. To send an IPC message, information about the originator and destination must be provided. This information is contained in the IPC key (the `ipc_key_t` structure). The IPC key contains the node ID, module ID, the application and process names, and the instance of an application or SINAP/SS7 subsystem, which are obtained from the application's registration parameters. Use the `ca_get_key()` function to obtain the `IPC_key` for a particular destination.

Other functions for handling IPC keys include:

- `ca_check_key()`
- `ca_swap_keys()`
- `ca_ascii_u32()`
- `ca_u32_ascii()`

Chapter 3
Application Design and Development

A *client application* is a C/C++ language program that includes calls to functions in the Common Application Services Layer (CASL) and can also include calls to the Integrated Services Digital Network User Part (ISUP) Services Support Library (ISSL). The CASL library contains C language functions that provide communications capabilities at the MTP, SCCP, and TCAP layers of the SS7 protocol. The CASL also provides services that the SINAP/SS7 management processes and client applications can use. It is the boundary between a client application and the SINAP/SS7 software. Both SINAP/SS7 management and client application processes logically reside on top of the CASL. The ISSL is a library of C language functions that provides communications capability at the ISUP layer of the SS7 protocol.

The CASL and ISSL libraries provide access to the MTP, SCCP, TCAP, and ISUP layers of the SS7 protocol. A client application uses the services of one of these layers and is considered a *user part* or *user* of that layer. For example, an application that interfaces with the SINAP/SS7 system at the MTP layer is considered an MTP user, just as an application that interfaces at the TCAP layer is considered a TCAP user part.

NOTE

The SINAP/SS7 system does not implement the Telephone User Part (TUP), but TUP can be implemented as part of an MTP user part application.

This chapter includes the following sections:

- "General Design Considerations" describes considerations of which you should be aware as you develop a SINAP/SS7 client application.
- "Considerations for Different Types of Applications" describes the major differences among the SINAP/SS7 network variant applications.
- "Developing Application Processing" provides background information that you might find useful as you develop your application's logic.
- "Activating/Deactivating a SINAP/SS7 Application" provides instructions for activating and deactivating client applications.
- "TCAP Client Applications" describes how to develop a TCAP client application.

- "SCCP Client Applications" describes how to develop an SCCP client application.
- "User Part (MTP) Client Applications" describes how to develop a user-part application, which interfaces to the MTP layer of the SS7 protocol.
- "Considerations for Implementing SINAP/SS7 Features" describes SINAP/SS7 features and provides instructions for implementing them in an application.
- "Error Handling" describes how to develop error-handling logic for your application.

General Design Considerations

As you develop applications to run on the SINAP/SS7 system, consider the following:

- The SINAP/SS7 system provides *archive libraries* and *shared object libraries*. The shared object libraries are also called *dynamic linked libraries* (DLL). You can link to either without modifying an application's programming interface.
- To use the services of the SINAP/SS7 platform, a client application must first register with the SINAP/SS7 system. At registration, the application defines its operating characteristics (for example, whether it will accept input at the MTP, SCCP, or TCAP layer and the type of primitives it will receive). By registering with the SINAP/SS7 system, an application also makes itself known to SINAP/SS7 node management. Thereafter, the SINAP/SS7 system has the information it needs to offer its services to that client application.
- The maximum number of SINAP/SS7 registered processes that can run concurrently is 256, including 29 SINAP/SS7 processes. Therefore, the SINAP/SS7 system can support a maximum of 227 application processes (including application instances) running concurrently. This value is defined by the variable, `MAX_APPL_OPC`, which is defined in the SINAP/SS7 `register.h` include file.
- The maximum number of applications that can be registered with the SINAP/SS7 system at any one time is 32. This value is defined by the variable, `MAX_APPL_SSN`, which is defined in the SINAP/SS7 `register.h` include file.
- UNIX does not notify SINAP/SS7 applications that an incoming message signaling unit (MSU) has arrived. It is the application's responsibility to call the CASL function `ca_get_msu()` or `ca_get_tc()` to determine whether an incoming MSU has arrived. (This is because the SINAP/SS7 system does not support a POLL or SELECT function.)
- Once an application has finished using SINAP/SS7 services, it must deregister by calling the CASL withdraw/terminate functions.
- If applications are configured to handle a large number of transactions, additional memory and heap space may be required (system tuning).

The remainder of this section discusses additional considerations of which you should be aware.

Multi-Threading Considerations (pthreads)

Generally all the SINAP functions mentioned in this document are **not** MT-SAFE. Generally the paradigm for developing SINAP applications should be as shown in this document for effective message handling, SINAP value added options (such as load control), and optimal performance. However, it is possible to use SINAP in a multi-threaded (pthreads) application in a perhaps sub-optimal manner after some careful considerations. Essentially, data to and from SINAP should be serialized to avoid MT-Safety problems. This may result in a loss of performance, since the powerful architectural functions of SINAP, such as load control, and the ability to have multiple SINAP application instances handling the data simultaneously across multiple processors, cannot be fully utilized.

You can use various techniques when you develop multi-threaded applications that utilize the CASL library SINAP function calls:

- Execute all SINAP function calls from the main thread only.
- Serialize all data flows to and from SINAP by using an intermediate queue mechanism.
- Wrap all SINAP calls with a common mutex.
- A combination of the above.

Note that, due to specifics of the SINAP implementation, the `ca_register()` function should only be called once and from the main thread only.

Figure 3-1. Example of mutex usage

```
pthread_mutex_t g_mutex;

...

pthread_mutex_lock(&g_mutex);

index = ca_alloc_tc();

pthread_mutex_unlock(&g_mutex);

...

pthread_mutex_lock(&g_mutex);

ca_put_tc(index);

pthread_mutex_unlock(&g_mutex);
```

Follow these recommendations in order to guarantee the fidelity of data provided to the application from SINAP, the fidelity of the data output by SINAP, or that SINAP will continue to operate correctly.

A multitreaded version of the `tcsend` sample program is provided in the `Samples/ccitt` directory. The name of the source file is `tcsend_mt.c` . The `tcsend_mt` program can be compiled by running '`make tcsend_mt`' in that directory. The sample `tcsend_mt.c` program utilizes the `pthread_create()` and `pthread_join()` POSIX thread library functions to create multiple threads and wait for threads' termination. The `send_route()` and `send_tcap()` program functions are executed within individual thread's context and call SINAP CASL library functions such as `ca_alloc_tc(), ca_put_tc(),` and `ca_dealloc_tc()`. It can be seen in the `tcsend_mt.c` source file that calls to the `ca_alloc_tc(), ca_put_tc()` and `ca_dealloc_tc()` SINAP CASL library functions are protected with the `pthread_mutex_lock()` and `pthread_mutex_unlock()` function calls. The multithreading technique demonstrated in the `tcsend_mt.c` sample program can be utilized by application developers.

Porting 32-Bit SINAP Applications to 64-Bit (HP-UX and Solaris only)

Most applications can remain in 32-bit mode on 64-bit systems. However, some applications, which manipulate very large data sets, are constrained by the 4GB address space limit in 32-bit mode. These applications can take advantage of the larger address space and larger physical memory of 64-bit systems. An enhanced 64-bit version of the SINAP/SS7 software supports 64-bit addressing and data files larger than 4GB in size. Existing 32-bit SINAP applications can be ported to the 64-bit mode to utilize these enhanced features. The SINAP/SS7 software for HP-UX or Solaris OS is designed in accordance with the 32-bit data model (ILP32) and the 64-bit data model (LP64). Some fundamental changes occur when moving from the ILP32 data model to the LP64 data model:

- `longs` and `ints` are no longer the same size
- `pointers` and `ints` are no longer the same size
- `pointers` and `longs` are 64 bits and are 64-bit aligned
- Predefined types `size_t` and `ptrdiff_t` are 64-bit integral types

These differences can potentially impact porting in the following areas:

- data truncation
- pointers
- data type promotion
- data alignment and data sharing
- constants
- bit shifts and bit masks
- bit fields
- enumerated types

The ANSI/ISO C standard specifies that C must support four signed and four unsigned integer data types: `char`, `short`, `int`, and `long`. There are few requirements imposed by the ANSI standard on the sizes of these data types. However, according to the standard, `int` and `short` should be at least 16 bits and `long` should be at least as long as `int`, but not smaller than 32 bits.

The 32-bit data model is called ILP32 because `ints`, `longs`, and `pointers` are 32 bits.

The 64-bit data model is called LP64 because `longs` and `pointers` are 64 bits. In this model, `ints` remain 32 bits.

The following table lists the basic C data types and their corresponding sizes in bits for both the ILP32 and LP64 data models:

Table 3-1. Data Type Size for ILP32 and LP64

C Data Type	ILP32 Size (Bits)	LP64 Size (Bits)
`char`	8	8
`short`	16	16
`int`	32	32
`long`	32	64
`long long`	64	64
`pointer`	32	64
`float`	32	32
`double`	64	64
`long double`	128	128
`enum`	32	32

Compiling 64-Bit Applications with 64-bit HP-UX OS

Stratus recommends that 64-bit SINAP application programs be compiled on the following platform:

- Continuum Series 400 systems with the PA-8500 or PA-8600 processor
- 64-bit HP-UX operating system (11.00.03)
- HP C/ANSI C Developer's Bundle for HP-UX 11.00 (B3901BA B.11.01.20)

To generate 64-bit object code for PA2.0 architecture, use `+DD64` HP C compiler option, which causes the macros `__LP64__` and `_PA_RISC2_0` to be `#defined`. This is the same as

`+DA2.0W`, but is the recommended option to use when compiling in 64-bit mode on the PA RISC2.0 architecture. In addition, the `+M2` option can be added to provide compilation warnings for possible problems connected with migration to the 64-bit mode. See Chapter 5, ‘‘Sample Applications,” for examples of compiling SINAP applications under 64-bit HP-UX OS.

NOTE

The `/usr/lib/pa20_64` is the default repository of the 64-bit archive and shared libraries (where `/usr/lib` is the default repository in 32-bit systems).

Compiling 64-Bit Applications with 64-bit Solaris OS

Stratus recommends that 64-bit SINAP application programs be compiled on the following platform:

- Sun Netra 20/T4[1] or SunFire V480 system
- 64-bit Solaris 8 operating system (Feb. 2002 version or later release)
- Sun Workshop Forte 6 C Compiler

To generate 64-bit object code for SPARC-V8 ISA, use `xarch=v9` compiler option, which predefines the `__sparcv9` macro and searches for v9 versions of lint libraries. In addition, the `xtarget=ultra2` specifies the target system (ultra2) for instruction set and optimization. See “Chapter 5, ‘‘Sample Applications,” for examples of compiling SINAP applications under 64-bit Solaris OS.

NOTE

The `/usr/lib/64` is the default repository of the 64-bit archive and shared libraries (where `/usr/lib` is the default repository in 32-bit systems).

Guidelines

As you develop applications to run on the 64-bit SINAP/SS7 system, consider the following guidelines.

Data Truncation

- Avoid Assigning longs to ints
- Avoid Storing Pointers in ints
- Avoid Truncating Function Return Values
- Use Appropriate Print Specifiers

[1] Note that Sun Microsystems refers to this model as either “Netra 20 Server” or “Netra T4”, but it is referred as “Netra 20/T4” in SINAP documentation to avoid the confusion.

Data Type Promotion

- Avoid Arithmetic between Signed and Unsigned Numbers

Pointers

- Avoid Pointer Arithmetic between longs and ints
- Avoid Casting Pointers to ints or ints to Pointers
- Avoid Storing Pointers in ints
- Avoid Truncating Function Return Values

Structures

- Avoid Using Unnamed and Unqualified Bit Fields
- Avoid Passing Invalid Structure References

Hardcoded Constants

- Avoid Using Literals and Masks that Assume 32 bits
- Avoid Hardcoding Size of Data Types
- Avoid Hardcoding Bit Shift Values
- Avoid Hardcoding Constants with malloc(), memory(3), string(3)

Tuning Your 64-bit Application

- Avoid performing mixed 32-bit and 64-bit operations, such as adding a 32-bit data type to a 64-bit type. This operation requires the 32-bit type to be sign-extended to clear the upper 32 bits of the register.
- Avoid 64-bit long division whenever possible.
- Eliminate sign extension during array references. Change `unsigned int`, `int`, and `signed int`, variables used as array indexes to `long` variables.

For additional information, see the references listed at next section.

References

For additional information see the following:

- *HP-UX 64-bit Porting and Transition Guide* (HP document # 5966-9887, June 1998)
- *HP-UX 64-bit Porting Concept* (http://devresource.hp.com/STK/64concepts.html)
- *HP-UX 64-bit compiler and linker changes* (http://devresource.hp.com/STK/64arch.html)
- *C User's Guide* (Sun WorkShop 6) (Sun Microsystems document # 806-3567, May 2000)
- *Data Size Neutrality and 64-bit Support* (http://www.UNIX-systems.org/whitepapers/64bit.html)

SINAP/SS7 Libraries

The SINAP/SS7 system offers both a shared object library, or dynamic linked libraries (DLLs), and an archive library. All SINAP/SS7 processes and sample programs use the SINAP DLL. The SINAP DLL allows you to update the following standard libraries using a patch release without performing a compile and link edit:

- Common Application Services Layer (CASL) Library
- ISUP Services Library (ISSL)

`LibCASL` libraries contain CASL functions and structures that simplify the development of service applications to be deployed in the SS7 network. `libissl` libraries contain ISUP services functions and data structures required to develop ISUP applications.

Library Type	Library Name	Location
DLL	`libCASL.sl` `libissl.sl`	`$SINAP_MASTER/Library` and `$SINAP_HOME/Library` The common locations for shared object libraries are `/usr/lib` in 32-bit systems, `/usr/lib/pa20_64` in 64-bit HP-UX system, and `/user/lib/64` in 64-bit Solaris system. A link is automatically installed during installation of the SINAP/SS7 software to `/usr/lib` DLLs.
Archive	`libCASL.a` `libissl.a`	`$SINAP_MASTER/Library` and `$SINAP_HOME/Library` (`$SINAP_HOME` root directory for your system installation)

NOTE

`$SINAP_HOME` represents the path of the currently installed and configured SINAP node, and `$SINAP_MASTER` represents the path to the master copy of all currently installed SINAP executables, libraries, and related files.

You can use either library. No application modifications are required to use the SINAP DLL. Select the library you want to use when you compile an application. For example, use a

command similar to the following to compile and link edit an application program with the SINAP DLL (`libCASL.so`) under HP-UX and Solaris:

```
@cc -I. -I$SINAP_HOME/Include -o [program name] [object name
list] -lCASL
```

Under the Stratus ft Linux operating system, the command is as follows:

```
@cc -I. -I$SINAP_HOME/Include -o [program name][object name
list] -lCASL -lLiS
```

This command assumes the shared object `libCASL.so` is located in `/usr/lib`.

NOTE

UNIX provides tools that are helpful when you are debugging an application. For the HP-UX operating system, you can use the following command to help debug an application:

For HP-UX operating systems, you can use the `chatr` command to view shared libraries.

For more information, see the man pages on your UNIX system.

To compile and link edit a SINAP/SS7 application using the archive library, use a command similar to the following to compile and link the program with the archive CASL library (`libCASL.a`) under HP-UX and Solaris:

```
@cc -I. -I$SINAP_HOME/Include -L /usr/ccs/lib -L /usr/lib -o
[program name] [object name list]
$SINAP_HOME/Library/libCASL.a
```

Under the Stratus ft Linux operating system, the command is as follows:

```
@cc -I. -I$SINAP_HOME/Include -L /usr/ccs/lib -L /usr/lib -o
[program name][object name list]
$SINAP_HOME/Library/libCASL.a -lLiS
```

The first release of the SINAP DLL will be part of a complete release version of the SINAP/SS7 system. Subsequent releases can be either complete releases or patch releases, which contain updates to the libraries. To install the complete SINAP/SS7 DLL release, you must compile and link edit the application. To install a patch release to update the libraries, you can simply stop and restart the user application and/or the SINAP node.

Client Application Models

Though no client application is provided with the SINAP/SS7 software, a client application is important to the design of the SINAP/SS7 system. The SINAP/SS7 system assumes that a client

application will behave as a queued-event *finite state machine* (FSM). An FSM is a program structure that behaves in a predictable manner, regardless of input. The predictability is achieved by processing an incoming event (or message) in exactly the same manner (within the application's current state) each time the application receives it. Memory of previous activity is achieved by switching the application between a finite set of states.

NOTE

While the SINAP/SS7 system provides no true client application, it does contain test applications for the purpose of validating the client application interface and function support. For example, the TCAP test application consists of two C programs called `tcsend.c` and `tcrecv.c`. The TCAP and other test applications are located in the directory `$SINAP_HOME/Samples/<network variant>`. For example, the test application for the CCITT network variant is located in the directory `$SINAP_HOME/Samples/ccitt`. For more information, see the sample applications in Chapter 5.

Control and Data Processes

A SINAP/SS7 client application can consist of one or more processes, each considered a *client application process*. An application can consist of a single process which performs both control and data processing or, it can consist of multiple processes, one of which performs control processing, and one or more that perform data processing.

NOTE

Unless otherwise noted, the term, *application,* is used throughout this chapter to refer to the application or application process that is executing a particular task (for example, calling a particular CASL function).

The manner in which an application process registers determines whether it is considered a control process, a data process, or both a control and data process.

- A *control process* is typically responsible for handling the application's management functions (for example, SS7 network management, interprocess communications (IPC), and the initialization and termination of data processes). An application can have only one control process.
- A *data process* is typically responsible for handling SS7 traffic, for example, reading inbound MSUs from the queue and responding with outbound MSUs. An application can have up to 16 separate data processes. Each data process is considered an instance of that application, or an *application instance*. When the process registers, the SINAP/SS7 system assigns it a unique instance ID.

For an application that consists of two or more processes, only one of the processes can send and receive SCCP management messages. This process is considered the application's *control process*. An application's control process must be registered to receive control primitives. The remaining processes (those that send and receive SS7 data) must be registered to receive data primitives only.

In the simplest case, a client application consists of a single process that receives and sends management messages. Such an application must be registered to receive both control and data primitives.

Single-Source SINAP/SS7 Code

All SINAP/SS7 network variants (ANSI, CCITT, TTC, NTT, and China) use the same source code. You specify which variant of the SINAP/SS7 system you want to run on your system when you install the SINAP/SS7 software. (See the *SINAP/SS7 Installation Guide* (R8060) for instructions.)

Previous to this single source code approach, there were separate versions for CCITT, ANSI, and Hybrid. If you have pre-Release 5.0 applications, you need not modify them to run with the consolidated source-code variant of the SINAP/SS7 system. To run existing applications, you need only recompile and rebind the applications with the new SINAP/SS7 libraries. However, due to differences between the older ANSI, CCITT, and Hybrid versions of the SINAP/SS7 system, you must run existing applications with the same variant of the SINAP/SS7 system for which the application was developed. For example, ANSI applications must be run on an ANSI installation, and CCITT applications must be run on a CCITT installation.

The SINAP/SS7 system uses several global variables to make the source-code consolidation invisible to you, the developer. For example, the `tblock.h` include file contains definitions for both types of dialogue- and transaction-handling structures: `tc_dhp_t` (used by CCITT/TTC/NTT/China applications) and `tc_thp_t` (used by ANSI applications). During installation, you specify which SINAP/SS7 network variant you want to run. The SINAP/SS7 system writes this information to the environment variable, `SINAP_VARIANT`, which serves as a pointer to the structures associated with that variant. Therefore, on a CCITT installation, the SINAP/SS7 system expects applications to use the `tc_dhp_t` structure, and on an ANSI installation, the SINAP/SS7 system expects applications to use the `tc_thp_t` structure.

NOTE

When you develop an application for a particular network variant of the SINAP/SS7 system, you must adhere to the programming and SS7 standards applicable to that variant. For example, if you are developing a CCITT application, you must use international 14-bit point-code addresses rather than the 24-bit point-code addresses used by ANSI applications.

By default, an application is compiled and bound with the variant of the SINAP/SS7 system that is currently running on your system. You can, however, override the default and compile and bind your application with another variant of the SINAP/SS7 system by making sure your application includes a reference to that variant's `variant.h` include file. For example, if you want to recompile a CCITT application and your system is currently running the ANSI variant of the SINAP/SS7 system, include a reference to the `ccitt_variant.h` include file in the CCITT application. Then, when you issue the command to recompile, the application is compiled with the CCITT variant of the SINAP/SS7 system and not ANSI.

UNIX Signal Remapping

The SINAP/SS7 system reassigns certain UNIX signals to different values. These new assignments do not interfere with normal client application process activities. Table 3-2 lists the UNIX signal and its reassignment. This information is contained in the `s7signal.h` include file. You must include `s7signal.h` in your application if it will use the SINAP/SS7 interprocess communications (IPC) signaling mechanism. In addition, the client application process must handle the `SIG_S7_IPC`, `SIG_S7_PF_BEGIN`, and `SIG_S7_PF_RIDETHRU` signals if it is registered for IPC signals or power-fail signals.

The include file `$SINAP_HOME/Include/s7signal.h` lists the UNIX-to-SINAP/SS7 signals and also references the UNIX file `signal.h`.

Table 3-2. UNIX-to-SINAP/SS7 Signal Remapping *(Page 1 of 2)*

UNIX Signal	SINAP/SS7 Remapping	Description
`SIGTTIN`	`SIG_S7_IPC`	Used for client processes requesting a signal when they receive an IPC message.
`SIGPOLL`	`SIG_S7_REROUTE`	Informs SINAP/SS7 processes that MTP management has updated the routing tables. The SINAP/SS7 system uses this signal internally.
`SIGALRM`	`SIG_S7_HIRES`	Used as a clock tick from SS7 driver for CASL deferred message delivery. The SINAP/SS7 system uses this signal internally.
`SIGTTOU`	`SIG_S7_PF_BEGIN`	Indicates that a power-failure event has started. The client process has 5 seconds before execution stops.

Table 3-2. UNIX-to-SINAP/SS7 Signal Remapping *(Page 2 of 2)*

UNIX Signal	SINAP/SS7 Remapping	Description
SIGXCPU	SIG_S7_PF_RIDETHRU	Indicates that power has been recovered before normal execution stopped. The client process can use this event to resume normal functioning. However, external peripherals (such as disks) might not yet have restarted upon receipt of this signal.

Because there can be only one occurrence of a signal outstanding, multiple IPC messages in the client application process' IPC message queue can result in fewer signals than messages. When the client application process receives the SIG_S7_IPC signal, it should read **all** messages from its queue. Because of their infrequency, SIG_S7_PF_BEGIN and SIG_S7_PF_RIDETHRU signals are not affected by multiple signals.

Tuning the Outgoing Batch Buffer Size

The SINAP/SS7 system provides the ability to improve performance by allowing you to change the size of the output batch buffer (CA_REG.batch_count). A batch buffer size of 1 causes the SINAP/SS7 internals to serialize all MSU transfers and is, therefore, very slow. By increasing the batch buffer size, overall MSU throughput can be increased. However, increasing the batch buffer excessively might result in congestion occurring (depending on the number of links). To provide the highest MSU throughput possible and prevent premature congestion, tune the value of CA_REG.batch_count with sufficient margin to allow the highest MSU throughput when links become inactive. Since the number of links depends on your configuration, Stratus provides no default recommendation for the value of CA_REG.batch_count.

Supporting Large Numbers of Transactions

The SINAP/SS7 system is capable of processing up to 300,000 transactions. However, in order to process large numbers of transactions, the size of the operating system's heap memory should be increased to 500 MB.

For example: to tune the operating system for approximately 500 MB of heap memory, the tunable parameters of the system should be set as follows:

Tunable Parameter	Suggested Value
HVMMLIM	24000000
SWMMLIM	24000000
SDATLIM	24000000

Tunable Parameter	Suggested Value
HDATLIM	24000000

For additional information on setting tunable parameters see, “Tunable Parameter Reconfiguration” in the *HP-UX Operating System: Administration Tasks* (B2355-90079).

TCAP EINTR Considerations

When you include a call to the `sleep()` function in a TCAP application, if the sleep interval is greater than one second, make sure to protect the `sleep()` function against an interrupted system call (EINTR) after the call to the SINAP `ca_register()` function.

To protect the function against an interrupted system call, use the method shown in the `samples/ccitt/tcsend.c` file. The method is implemented in the `delay_counter` function in that file. In other words, a `sleep(1)` function should be surrounded by a definite loop that loops for the number of seconds required by the `sleep()` call.

When writing a TCAP application that requires input from the keyboard after the call to the SINAP `ca_register()` function , instead of using the `(void) scanf()` function, use the `READ_MENU_ITEM` helper macro, defined in `ca_glob.h`. See this macro definition for example processing.

Generally any blocking calls done after `ca_register()` function should be protected as necessary against EINTR. See the discussion of implementing the `sleep()` and `scanf()` functions above.

At startup and before calling the `ca_register()`function, the application should gather any static information, information that does not change while SINAP processing is in progress. This practice will avoid the need for `scanf()` later on.

Considerations for Different Types of Applications

This section describes the things you must consider as you develop applications that interface with the SINAP/SS7 system at each of the different SS7 protocol levels (MTP, SCCP, TCAP, and ISUP). It presents a list of the include files required by different types of SINAP/SS7 applications and describes the differences between the ANSI, CCITT, TTC, NTT, and China variants of the SINAP/SS7 system.

Include Files Required for Different Types of Applications

The following is a list of the include files that different types of applications must reference. For an application to reference an include file, it must contain an `#include` statement for that file. For example, the statement `#include <tblock.h>` references the SINAP/SS7 `tblock.h` include file. (For a description of the SINAP/SS7 include files, see the section, “SINAP/SS7 Include Files” in Chapter 2.)

The include file `caslinc.h` references many of the include files required by different types of SINAP/SS7 applications (for example, `caslinc.h` references `ca_error.h`, `arch.h`, `sinap.h`, `register.h`, `iblock.h`, `mblock.h`, `tblock.h`, and `sinapintf.h`). Therefore, the following list mentions those include files that are not referenced by `caslinc.h`.

- An application that is registered to receive control primitives should reference the include file `prims3.h`, which is **not** referenced by `caslinc.h`. This applies to applications that register to receive only control primitives and applications that register to receive both control and data primitives.
- An SCCP application that handles SS7 traffic (that is, accepts inbound MSUs and/or processes responses in outbound MSUs) must reference the include files `sccphdrs.h` and `sccp-intrn.h`, which are **not** referenced by `caslinc.h`.
- In addition to `caslinc.h`, an application that interfaces with the SINAP/SS7 system at the TCAP boundary must reference the include files `tcap.h` and `scmg-prims.h`, which are required so the application can use SCCP primitives.
- An ISUP services feature application must reference `issl.h`, which incorporates all the include files necessary for the application to use the ISSL. For more information, see the *SINAP/SS7 ISDN User Part (ISUP) Guide* (R8053).
- In addition to the include files required by all SINAP/SS7 applications, an application that implements load control must reference the include file, `locon.h`, which is already referenced in `caslinc.h`.
- Application processes that send and receive IPC messages (such as an application's control process) must reference the include files `blkhdr.h`, `timestamp.h`, `iblock.h`, and `ipctbl.h`, each of which is referenced in `caslinc.h`.
- To implement BITE monitoring, an application process must reference the IPC-related include files listed in the preceding paragraph and also the include files `bitemon.h` and `measure.h`, which are referenced in `caslinc.h`.
- To log events to the trouble treatment table, an application process must reference the include files `event.h`, `event3.h`, and `treatment.h`, in addition to the IPC-related include files listed above. (All of these files are referenced in the `caslinc.h` include file.)

Network Variant Differences

This section describes the differences among the network variants in the SINAP/SS7 system. You should consider these differences as you design and develop applications to run on the SINAP/SS7 system. The section, "Considerations for Implementing SINAP/SS7 Features," later in this chapter, presents additional considerations. Throughout the manual, additional differences are noted where applicable. Some basic differences between the network variants are described in the following list:

- The network variants of the SINAP/SS7 system use different formats for specifying point codes (for example, for an SCCP called or calling-party address).

- In CCITT, point codes are defined using 14-bit binary code in the range of 0 through 16383, for example, 5674.
- In TTC and NTT, point codes are defined as 16-bit binary codes in the range of 1 through 65536, for example, 10005.

  Due to the differences in point code lengths, the MTP routing label used for TTC and NTT MSUs is 36 bits, 4 bits longer that the MTP routing label for CCITT MSUs.

- In ANSI, point codes are defined using 24-bit binary codes divided into three fields of 8-bit components separated by hyphens and arranged in the format *network-cluster-member* (where *network* is the ID of the network to which the node belongs, *cluster* is the ID of the cluster to which the node belongs, and *member* is the ID of the node).

  – For *network*, specify a decimal value in the range of 1 through 254.
  – For *cluster*, specify a decimal value in the range of 1 through 255.
  – For *member*, specify a decimal value in the range of 1 through 255.

  An example of a point code for the ANSI network variant is 42-135-6.

- In China, the point codes are defined in almost the same way as ANSI variant point codes. They also use a 24-bit binary code, divided into three fields of 8-bit components separated by hyphens in the format *network-cluster-member* (where *network* is the main signaling area, *cluster* is the sub-signaling area, and *member* is the signaling point code). The values for each component are different than those for the ANSI point codes. The valid range you can specify for the network, cluster, and member components is 0 through 255, with one restriction. The SINAP/SS7 system reserves the point code 0-0-0 for internal system use. All other point code combinations within the 0 through 255 range are valid.

  A sample China point code is 1-22-111.

  This manual refers to China point codes in terms of the ANSI equivalents, network, cluster, member.

- When you use the TTC and NTT network variants, you can specify whether the application or the CASL is responsible for sending user-in-service (UIS) and user-out-of-service (UOS) messages to SCMG. This is done by specifying the environment variable `TTC_WITH_NSTATE`. For more information, see Activating/Deactivating a SINAP/SS7 Application later in this chapter.
- In the CCITT network variant, the SINAP/SS7 system supports the ITU-T 1993 SCCP management (SCMG) procedures. (SCMG messaging is handled by UDT only, therefore, XUDT messages are discarded.) However, an environment variable (`CCITT_XUDT_SCMG`) can be set to enable the SINAP/SS7 system to respond with a subsystem allowed (SSA) message when a subsystem test (SST) message is sent via XUDT.

- The MTP, SCCP, TCAP, and ISUP timers have different identities, default values, and ranges of values depending on the variant of the SINAP/SS7 system you are using.
- Signaling point codes in the NTT network variant, like TTC point codes, are defined as 16-bit binary codes. All point codes must be within the number range 1-65536 (for example, 1005).

  In the NTT network variant, the 16-bit point codes are divided into three components separated by hyphens and arranged in the format `M-S-U` (similar to the ANSI point code format):

  —`M = Main number area` - A 5-bit code (a decimal value in the range 0-31 that identifies the main number area to which the node belongs.

  —`S = Subnumber area` - A 4-bit code (a decimal value in the range 0-15) that identifies the subnumber area to which the node belongs

  —`U = Unit number` - A 7-bit code (a decimal value in the range 0-127) that defines the node ID.

  Incoming TFP, TFR, and TFA messages can specify the following three types of destinations:

  —`Mxx`

  —`M-Sx`

  —`M-S-U`

  The `x` character represents a *wildcard* that can be used to specify *wide-ranging* destinations such as subnumber area. These wide-ranging destinations points can specify multiple destination points (up to 13) as *affected destinations* if the value specified in the message's Coding field is `0000` or `0001`. When the Coding field value is 0010 - 1111, the node receiving the TFP, TFR or TFA message processes the unique point code that matches the M-S-U code specified in the message.

  `If th`e receiving node finds no matching `Mxx`, `M-Sx`, or `M-S-U` codes configured, the node issues an error message.

  For outbound route set test (RST) messages sent in response to TFP, TFR, or TFA messages, the message's Coding field contains the value 0010 which defines a specific point code.

Configuration Requirements and Limitations

Table 3-3 lists the requirements and limitations for the basic SINAP/SS7 configuration parameters and the features that are or are not supported by the different network variants.

Table 3-3. Configuration Requirements and Limitations *(Page 1 of 9)*

Item	Limitation or Support
Maximum number of signaling links	128 per SINAP module. **The total for all nodes must not exceed 128**. **Note:** You can only configure 128 links on the Continuum Series 400 systems equipped with PA-8500 or PA-8600 processors.
Maximum IOA cards per Continuum Series 400 & Series 400-CO systems with PA-8000 processor: PA-8500 and PA-8600 processors:	 U403 = 8 cards (4 Links per card) U420 = 4 cards (8 Links per card) U403 = 8 cards (4 Links per card) U420 = 8 cards (8 Links per card) U916 = 8 cards (32 Links per card)
Maximum IOA cards per Netra 1400 Series systems	U915 = 2 cards (32 Links per card)
Maximum IOA cards per Netra 20/T4 system	U915 = 3 cards (32 Links per card)
Maximum IOA cards per ftServer T30 Series system	U918 = 4 cards (32 Links per card)
Maximum IOA cards per ftServer T50 Series system	U918 = 10 cards (32 Links per card)
Link operating speeds supported: CCITT/ANSI/China TTC/NTT	 4.8, 9.6, 19.2, 38.4, 56, 64 kbit/s If you are creating an ARTIC synchronous link, you must specify a link speed of 0 if you connect to a modem or other device that provides external clocking. Supports baud rates of 4800, 48,000, or 64,000.
Maximum number of link sets	16 per node (a module with 4 nodes can have a maximum of 64)
Maximum number of links per link sets	16

Table 3-3. Configuration Requirements and Limitations *(Page 2 of 9)*

Item	Limitation or Support
Maximum number of routes per route set: CCITT/TTC/NTT/China ANSI	 8 4
Maximum number of route sets	2048 per node (a module with 4 nodes can have a maximum of 8192)
Maximum number of load-shared routes	2
Load sharing for route sets supported: CCITT TTC NTT ANSI China	 Yes Yes Yes No Yes
Maximum number of destination point codes (DPCs)	2048 per node (a module with 4 nodes can have a maximum of 8192)
Maximum number of DPCs reachable by one link set	2048 per node (a module with 4 nodes can have a maximum of 8192)
Maximum number of concerned point codes (CPCs)	64 per local SSN (Up to 512 can be accommodated with a special patch. Contact the CAC for more information.) **Note:** The TTC and NTT network variants support CPCs only if the environment variable `TTC_WITH_NSTATE` is defined.
Maximum number of duplicate concerned point codes (DUCPCs): CCITT/ANSI/China TTC/NTT	 1 per local SSN Not supported
Distributed logical point codes (DLPCs) supported: CCITT/ANSI/China	 Yes, if DLPC is configured on the SINAP node (via `/etc/config_sinap` script)
Maximum number of logical point codes (LPCs) allowed for registered processes CCITT/ANSI/China	 16 per node in addition to own signaling point code (only if DLPC feature is configured on the node via `/etc/config_sinap` script)

Table 3-3. Configuration Requirements and Limitations *(Page 3 of 9)*

Item	Limitation or Support
Application failure detection with notification to backup DLPC application supported: CCITT/ANSI/China	 Yes (only if DLPC feature is configured on the node via `/etc/config_sinap` script)
Maximum number of applications registered with SINAP	32 per node (a module with 4 nodes can have a maximum of 128)
Maximum number of processes registered with the SINAP node that can run concurrently	255 per node
Instances per application	16 instances per application
`drda_daemon` processes	1 per node (a module with 4 nodes can have a maximum of 4)
Maximum number of links per combined link set (CLS) ANSI CCITT/TTC/NTT/China	 32 links (2 link sets) per CLS Not supported
Combined link sets supported: CCITT TTC NTT ANSI China	 No No No Yes - 4 per node (a module with 4 nodes can have a maximum of 16) No
Signaling Connection Control Part (SCCP) message modes supported: CCITT TTC NTT ANSI China	**Note:** Connectionless: Class 0 (unsequenced) and Class 1 (sequenced), Connection-oriented: Class 2 and Class 3 Class 0, 1, 2, 3 Class 0, 1 Class 0, 1 Class 0, 1 Class 0, 1, 2, 3
XUDT and XUDTS message handling supported: CCITT TTC NTT ANSI China	 Yes No No No Yes

Table 3-3. Configuration Requirements and Limitations *(Page 4 of 9)*

Item	Limitation or Support
SCCP addresses supported for destination point code (DPC) and originating point code (OPC): CCITT TTC NTT ANSI China	 14-bit point code format 16-bit point code format 16-bit point code format 24-bit point code format 24-bit point code format
Global title translation (GTT) supported: CCITT TTC NTT ANSI China	 Yes Yes Yes Yes Yes
Partial GTT supported: CCITT/TTC/ NTT / ANSI / China	 Yes, if the environment variable `PARTIAL_GTT` is defined.
SCCP backup routing for GTT only supported: CCITT TTC NTT ANSI China	 Yes, if the environment variable `GTT_WITH_BACKUP_DPC_SSN=1` is defined. No No No No
Global title (GT) addressing supported: CCITT TTC NTT ANSI China	 Yes Yes Yes Yes Yes
Connection-oriented features (COF) supported: CCITT TTC NTT ANSI China	 Yes No No No Yes

Table 3-3. Configuration Requirements and Limitations *(Page 5 of 9)*

Item	Limitation or Support
Number of link-congestion levels supported: CCITT and China	Three optional congestion levels: • International one congestion onset and one congestion abatement • National multiple states with congestion priority option • National multiple congestion states without congestion priority (Default is international one congestion onset and one congestion abatement if no environment variable is set to define link congestion levels)
ANSI, TTC, and NTT	National multiple congestion states with congestion priority automatically implemented
ISUP service features supported: CCITT, NTT, China, and ANSI	Yes, if `ISUP_FEATURE` environment variable is defined (see the *SINAP/SS7 ISDN User Part (ISUP) Guide* (R8053)
TTC	No
MTP signaling point restart supported: CCITT and China	Yes, if `MTP_WHITE_BOOK_RESTART` environment variable is defined. Default is CCITT 1988 network processing.
TTC NTT	No No
ANSI	Yes, if `MTP_ANSI92_RESTART` environment variable is defined. Default is ANSI 1990 network processing with no restart processing.
MTP user part unavailable (UPU) messages and user flow control (UFC) supported: CCITT TTC NTT ANSI China	 Yes No No Yes Yes

Table 3-3. Configuration Requirements and Limitations *(Page 6 of 9)*

Item	Limitation or Support
Fictitious originating point code (FOPC) supported: CCITT TTC NTT ANSI China	 No No No Yes No
Signaling network messages (SNM) with non-zero SLCs supported: CCITT/China TTC NTT ANSI	 Yes (`MTP_WHITE_BOOK_SLC` environment variable must be defined) No No Yes (no need to set an environment variable)
MTP time-controlled changeover (TCCO) supported: CCITT/China TTC/NTT ANSI	 Yes, CCITT 1988 TCCO is the default. To implement CCITT 1993 TCCO processing, define the environment variable `MTP_WHITE_BOOK_TCCO`. Yes (automatically implemented) Yes, ANSI 1990 TCCO is the default. (Implemented automatically if MTP restart is enabled; otherwise, you must set the environment variable `MTP_ANSI92_TCCO`)
Time-controlled diversion (TCD) supported: CCITT/TTC/NTT/China ANSI	 Yes (Implemented automatically; no environment variable must be set) Yes (Implemented automatically if MTP restart is enabled; otherwise, you must set the environment variable `MTP_ANSI92_TCD`)

Table 3-3. Configuration Requirements and Limitations *(Page 7 of 9)*

Item	Limitation or Support
Remote processor outage control (POC) supported: CCITT TTC NTT ANSI China	 Yes No No Yes Yes
Local processor outage control (POC) supported: CCITT TTC NTT ANSI China	 No No No No No
Preventive cyclic retransmission (PCR) supported: CCITT TTC NTT ANSI China	 No No No No No
International network indicator supported: CCITT TTC NTT ANSI China	 Yes Yes Yes No Yes
Loopback detection supported: CCITT TTC NTT ANSI China	 Yes, if the environment variable `LOOPBACK_DISPLAY` is set. No No No No

Table 3-3. Configuration Requirements and Limitations *(Page 8 of 9)*

Item	Limitation or Support
Transfer-restricted message handling supported:	
CCITT	Yes, if the environment variable `MTP_WHITE_BOOK_TFR` is defined.
TTC	No
NTT	No
ANSI	Yes, implemented automatically (according to 1992 ANSI Standards); no environment variable needs to be set. To configure the node according to the1988 ANSI standards, set the environment variable `MTP_ANSI88_RSR_RST`. See the section, “RSR/RSP in Response to TFR/TFP,” later in this chapter for more information.
China	No
Even distribution of messages by routing solely on SLS and DPC supported:	
CCITT and China	Yes, if the `MTP_SLS4_LOAD_SHARE` environment variable is defined before starting or restarting the SINAP/SS7 system.
TTC, NTT, and ANSI	No
Selection of 5-bit or 8-bit Signaling Link Selection (SLS) processing schemes for all incoming and outgoing traffic supported:	
CCITT/TTC/NTT/China	No
ANSI	Yes (Specified using the `CHANGE_SLSTYPE` command; default is 5-bit processing)
Custom Application Distribution (CAD) supported:	
CCITT	Yes
TTC	No
NTT	No
ANSI	Yes
China	No

Table 3-3. Configuration Requirements and Limitations *(Page 9 of 9)*

Item	Limitation or Support
Maximum number of ServiceKey values per application (CAD): ANSI/CCITT TTC/NTT/China	 64 No
Maximum number of ServiceKeys supported for a specific SSN/OPC criteria (CAD): ANSI/CCITT TTC/NTT/China	 256 No
Maximum number of fallback applications supported for a specific SSN/OPC criteria (CAD): ANSI/CCITT TTC/NTT/China	 1 (Value specified for ServiceKey must be 0) No
Maximum number of SSNs per application (Enhanced Distribution and CAD): ANSI/CCITT TTC/NTT/China	 32 No
Maximum number of OPCs per application (Enhanced Distribution and CAD): ANSI/CCITT TTC/NTT/China	 128 No

Structure Differences

Several CASL structures have variant-specific versions. For example, the structure `msu_t` has several variant-specific versions: `ccitt_msu_t`, `ansi_msu_t`, and `ttc_msu_t`. The China variant uses the `ansi_msu_t` structure and the NTT variant uses the `ttc_msu_t` structure. When you develop an application, you can code the application so it uses the generic version of a CASL structure (for example, `msu_t`) or a particular variant-specific version (for example, `ttc_msu_t`). The global variable `SINAP_VARIANT`, which defines the network variant executed on the SINAP node, links the generic versions of CASL structures to their corresponding variant-specific versions, thus making the source code consolidation invisible to programmers.

TTC-specific versions of existing CASL structures are described in the following chart. TTC structures are defined in the include file `$SINAP_HOME/Include/mblock.h`.

Generic Structure	TTC Structure
`msu_t`	`ttc_msu_t`
`sccp_user_t`	`ttc_sccp_user_t`
`snm_user_t`	`ttc_snm_user_t`

In the CCITT/TTC/NTT/China variants of the SINAP/SS7 system, the `tc_dhp_t` structure is used to communicate dialogue-handling information.

In the ANSI variant of the SINAP/SS7 system, the `tc_thp_t` structure is used to communicate transaction-handling information.

The format of the `dest_addr` and `orig_addr` fields in the `tc_dhp_t` structure and the `tc_thp_t` structure differ. The `dest_addr` field defines the SCCP called party address, or destination point code (DPC), of a TCAP component and the `orig_addr` field defines the SCCP calling-party address, or originating point code (OPC), of the component.

Code the `dest_addr` and `orig_addr` fields according to the recommendations for the type of application you are developing.

- For CCITT, TTC, NTT, and China applications, refer to ITU-T (CCITT) Recommendation Q.713.
- For ANSI applications, refer to ANSI Recommendation T1.112.3, *SCCP Formats and Codes*.

Differences in CASL Functions Supported

There are minor differences in the CASL functions supported by the CCITT, ANSI, TTC, NTT, and China network variants that run on the SINAP/SS7 system. The following list describes the differences:

- In the CCITT/TTC/NTT/China variants of the SINAP/SS7 system, communication between two TCAP applications is considered a *dialogue*. To initiate a dialogue, an application must call the function, `ca_get_dial_id()`, to retrieve a unique ID to assign to the dialogue. The function `ca_rel_dial_id()` releases it.
- In the ANSI variant of the SINAP/SS7 system, communication between two TCAP applications is considered a *transaction*. To initiate a transaction, an application must call the function, `ca_get_trans_id()`, to retrieve a unique ID to assign to the transaction. The function `ca_rel_trans_id()` releases it.

Primitives Supported

The following list presents the differences between the primitives supported in the different variants of the SINAP/SS7 system.

- In the CCITT, TTC, NTT, and China network variants of the SINAP/SS7 system, the TCAP primitive `TC_BEGIN` is used to initiate a dialogue.

  In the ANSI network variant of the SINAP/SS7 system, the TCAP primitives `TC_QRY_W_PERM` and `TC_QRY_WO_PERM` are used in place of `TC_BEGIN`.

- In the CCITT, TTC, NTT, and China network variants of the SINAP/SS7 system, the TCAP primitive `TC_CONTINUE` is used to continue a dialogue.

  In the ANSI network variant of the SINAP/SS7 system, the TCAP primitives `TC_CONV_W_PERM` and `TC_CONV_WO_PERM` are used in place of `TC_CONTINUE`.

- In the CCITT, TTC, NTT, and China network variants of the SINAP/SS7 system, the TCAP primitive `TC_END` primitive is used to end a dialogue.

  In the ANSI network variant of the SINAP/SS7 system, the TCAP primitives `TC_RESPONSE` and `TC_NO_RESPONSE` are used in place of `TC_END`.

- In the CCITT, TTC, NTT, and China network variants of the SINAP/SS7 system, the TCAP primitive `TC_INVOKE` is used to request services from another TCAP user.

  In the ANSI network variant of the SINAP/SS7 system, the TCAP primitives `TC_INVOKE_L` and `TC_INVOKE_NL` are used in place of `TC_INVOKE`.

Developing Application Processing

A typical SINAP/SS7 client application is designed to interface with other applications in the SS7 network. To use the services of the SINAP/SS7 platform, a client application process must first register with the SINAP/SS7 system. At registration, the application process defines its operating characteristics and makes itself known to node management, the SS7 network, other processes within the application, and other applications. Once the application process is registered, the SINAP/SS7 system has the information it needs to offer its services to that client application.

One of the operating characteristics that an application process specifies at registration is a unique ID by which the SINAP/SS7 system can reference the application process. TCAP and SCCP applications use a specified subsystem number (SSN). MTP user-part applications (such as TUP) use a service information octet (SIO).

The remainder of this section describes the tasks a SINAP/SS7 client application performs to communicate with another application:

- Register with the SINAP/SS7 system.
- Go into service in the SS7 network.

- Handle SS7 messages.
- Go out of service.
- Withdraw from the SS7 network.

In addition, the application can also perform the following tasks:

- Send MML commands to another process.
- Monitor and intercept SS7 messages for debugging purposes.
- Auto-start BITE monitor processes.
- Debug processing logic.
- Report events to the trouble management process.
- Perform health-check operations.

In addition, the section, Activating/Deactivating a SINAP/SS7 Application provides instructions for activating and deactivating SINAP/SS7 client applications.

Registering with SINAP/SS7

To use the services of the SINAP/SS7 platform, a client application must first register. Registering with the SINAP/SS7 system makes the application known to node management, the SS7 network, other processes within the application, and other applications. When an application registers, the SINAP/SS7 system allocates space for that application in *shared virtual memory*, attaches shared memory tables to the application, and opens the SS7 device.

NOTE

If a client application is composed of a number of processes, **each** process must register separately with the SINAP/SS7 system.

To register with the SINAP/SS7 system, an application process initializes the `register_req_t(CA_REG)` structure to define its operating characteristics. The application then calls the `ca_register()` function and passes it the `register_req_t (CA_REG)` structure. For instructions on how to register your client application with the SINAP/SS7 system, see the appropriate section later in this chapter that provides instructions for developing a particular type of application.

Going Into Service

Once registered with the SINAP/SS7 system, the application can set signal handlers to handle the SINAP signals, such as `SIG_S7_PF_BEGIN` and `SIG_S7_PF_RIDETHROUGH`. Then the application must inform SCCP management that it is ready to begin processing. For the CCITT, ANSI, and China network variants, the application sends an `N_STATE`

`User-In-Service` (UIS) primitive to SCCP management via IPC. If the application consists of several processes, the application's control process (the process that receives and sends SCCP management messages) is responsible for sending the `N_STATE User-In-Service` primitive to SCCP management (SCMG). For information about the `N_STATE` primitive and its formats, see the `$SINAP_HOME/Include/scmg-prims.h` include file.

NOTE

If a concerned point code (CPC) was configured for the application, the SINAP/SS7 system notifies that CPC that the application's status has changed and the application is now available. The CPC can then start sending MSUs to the application. (See the *SINAP/SS7 User's Guide* (R8051) for information about CPCs.)

For the TTC and NTT network variants, after an application registers, the application must send a UIS message to SCMG to indicate that it is ready to begin processing. In addition, before withdrawing from the SS7 network and terminating completely, an application must send a user-out-of-service (UOS) message to SCMG to indicate that it wants to stop processing.

For TCAP and SCCP applications, you can define the environment variable `TTC_WITH_NSTATE` to specify who is responsible for sending the UIS and UOS messages (`N_STATE_REQ SCMG_UIS` and `N_STATE_REQ SCMG_UOS`, respectively) to SCMG.

- If you do not define the `TTC_WITH_NSTATE` environment variable, the CASL automatically sends UIS and UOS messages on behalf of the application.

  NOTE

  According to the TTC JT-Q recommendations, this is the preferred method of handling UIS/UOS messages.

  A UIS message is sent to SCMG when the application calls the `ca_register()` function to register; a UOS message will be sent to SCMG when the application calls the `ca_withdraw()` function to withdraw from the SS7 network. If `TTC_WITH_NSTATE` is not defined, SCMG ignores UIS and UOS messages sent by the application.

- If you define the `TTC_WITH_NSTATE` environment variable, the application is responsible for sending UIS and UOS messages to SCMG. For an example of how to code your application to send these messages, see the functions `send_nstate_uis_to_sccp()` and `send_nstate_uos_to_sccp()` in the TCAP sample program `tcsend.c` or in the SCCP sample program `scsend.c`. (Both sample programs are located in the `$SINAP_HOME/Samples/ttc` directory.)

NOTE

You must define the `TTC_WITH_NSTATE` environment variable at a UNIX prompt **before** you start the SINAP/SS7 system. You need not assign a value to the variable; the SINAP/SS7 system simply verifies that the variable exists. For instructions on how to define environment variables, see Appendix B.

Handling SS7 Messages

The CASL provides a number of functions for handling messages at each of the different SS7 protocol layers. The following list presents the message-handling functions available for different types of applications. Detailed instructions for how an application handles SS7 messages are provided later in this chapter.

For SCCP and MTP applications, the CASL provides the following functions:
`ca_get_msu()` retrieves an incoming MSU, which contains an SS7 message.

- `ca_put_msu()` sends an outgoing MSU to the SS7 network.
- `ca_flush_msu()` sends the MSUs in the outgoing buffer to the SS7 network, regardless of whether the buffer is full. This function is also available to TCAP applications.

For TCAP applications, the CASL provides the following functions:

- `ca_get_tc()` retrieves an incoming TCAP component from another TCAP client application.
- `ca_put_tc()` sends an outgoing TCAP component to another TCAP client application.
- `ca_process_tc()` controls the processing of TCAP components.

TCAP applications also use the following CASL functions to manage the `T_Blocks` that contain the TCAP components included in TCAP SS7 messages: `ca_alloc_tc()`, `ca_dealloc_tc()`, `ca_get_trans_id()`, and `ca_rel_trans_id()`.

Sending MML Commands

The CASL provides two functions for sending and responding to MML commands: `ca_put_cmd()` and `ca_put_reply()`.

A client application process sends commands and receives replies using IPC messages. To send a command to any registered SINAP/SS7 process, the client application process uses the `ca_put_cmd()` function. The `ca_put_cmd()` function formats an IPC message and sends the message to its destination, based on information in the IPC key. The receiving process responds using the `ca_put_reply()` function. This function formats a response and forwards it to the originating process.

Monitoring and Intercepting SS7 Messages

The `ca_enable_mon()` and `ca_disable_mon()` functions allow a client application process to enable or disable input and output buffer monitoring from one or more of its processes. Though the Terminal Handler provides monitoring capabilities during registration, these functions allow monitoring based on events that can be conditionally coded or dynamically activated.

The intercept functions, `ca_enable_intc()` and `ca_disable_intc()`, allow an application to enable or disable the intercept mode. An application uses the intercept mode to run network simulation, with the Built-In Test Environment (BITE) subsystem performing the actual simulation.

- The `ca_enable_intc()` function tells the BITE subsystem to place the SS7 communication path to the calling client application process in intercept mode, and starts a scenario execution program under the control of BITE to simulate network activity. The scenario execution results are logged. You can enable this feature by issuing the MML `START-MON` command or by having the application register with specific flags set in its registration parameter structure. For instructions, see the section, Auto-Starting BITE Monitor Processes later in this chapter.
- The `ca_disable_intc()` function tells BITE to discontinue scenario execution operations and to return the process to normal network communications activity.

Auto-Starting BITE Monitor Processes

If you want the SINAP/SS7 system to automatically initiate a BITE monitor process when your application starts, code the application so it registers with the SINAP/SS7 system with the following values assigned to these `register_req_t` (`CA_REG`) structure fields.

- If `fmon_ss7` is set to the value `1`, the SINAP/SS7 system initiates a BITE monitor process to monitor all of the application's SS7 processes.
- If `fmon_ipc` is set to the value `1`, the SINAP/SS7 system initiates a BITE monitor process to monitor all of the application's IPC activities.
- Initialize (`CA_REG`) `mon_filename` to the name of the file to which you want the BITE monitor messages logged.

You are responsible for disabling BITE monitor processes. You can do this by issuing the MML commands, `DISPLAY-MON` (to determine the ID of the monitor process), followed by `STOP-MON`. Or, you can have the application call the function `ca_disable_mon()`.

NOTE

You can issue the MML `STOP-MON` command while the application is running or after it terminates. However, if the application's registration parameters are set to automatically initiate a BITE monitor process when the application is

activated, you must reset the application's `fmon_ss7` and `fmon_ipc` registration parameters to `0` before restarting the application. Otherwise, when the application is restarted, the SINAP/SS7 system will automatically initiate a new monitor process.

Debugging Processing Logic

The `ca_dbg_display()` and `ca_dbg_dump()` functions provide debugging capabilities. The `ca_dbg_display()` function traces various points of a message flow; `ca_dbg_dump()` displays specific memory segments.

Reporting Events

The `ca_put_event()` function provides the SINAP/SS7 system with an event-reporting capability. A client application process can use `ca_put_event()` to report status or alarms to the trouble management process in the node management subsystem.

Health-Check Operations

The CASL provides the following two health-check functions for determining the status of a client application process:

- `ca_health_chk_req()` sends health-check messages.
- `ca_health_chk_resp()` responds to health-check messages.

If a client application process is registered for health checks, node management sends the process a health-check message at fixed time intervals. This time interval is defined by the SINAP/SS7 environment variable, `SINAP_HEALTH_INTERVAL` (seconds). To pass the health check, the client application process must reply within the time period defined by the SINAP/SS7 environment variable, `SINAP_HEALTH_TIMEOUT` (seconds). If the client application process does not respond to two sequential health-check messages, the node management subsystem declares the process failed. It then performs any failure processing that the application specified at registration.

Going Out of Service

For an application to go out of service, the application's control must send an `N_STATE User-Out-of-Service` primitive to SCCP management (via IPC). When SCCP management receives this message, it sets the state of the application to `UNAVAILABLE`. Thereafter, when the SINAP/SS7 system receives an incoming SS7 MSU destined for the application, the MSU is discarded or returned to the sender (`Return-On-Error`). MSUs that the SINAP/SS7 system had already received are forwarded to the client application. If the application has outbound MSUs waiting on the queue, the SINAP/SS7 system routes these MSUs to their destinations.

NOTES

1. If a CPC was configured for the application, the SINAP/SS7 system notifies that CPC that the application's status has changed and the application is no longer available. The CPC uses this information to determine whether to continue sending MSUs to the application. See the *SINAP/SS7 User's Guide* (R8051) for information about concerned point codes.
2. If the application is associated with a duplicate concerned point code (DUCPC), the application must notify that DUCPC that it intends to go out-of-service. The DUCPC can then activate its copy of the application to take over processing so that service is not disrupted while this application is out-of-service. See the *SINAP/SS7 User's Guide* (R8051) for information about duplicate concerned point codes.

Withdrawing From the SS7 Network

An application can call the `ca_withdraw()` function to gracefully terminate processing. This function causes the SINAP/SS7 system to remove the application from inbound load distribution. This allows the application to continue processing existing messages, but prevents new messages from being routed to the application.

After calling the `ca_withdraw()` function, an application is still registered with the SINAP/SS7 system. To completely halt processing and remove its association with the SINAP/SS7 system (to de-register), the application should call `ca_terminate()`.

Activating/Deactivating a SINAP/SS7 Application

The procedures in this section provide information about how to activate and deactivate SINAP/SS7 client applications. The section, Developing Application Processing described the procedures a client application must perform. This section describes the procedures you or your system administrator must perform outside the client application.

Activating a SINAP/SS7 Client Application

Once the SINAP/SS7 system has successfully started, you can use any one of the following methods to start a SINAP/SS7 client application:

- The startup script $SINAP_HOME/Bin/startappl allows client applications to start automatically when you start the SINAP/SS7 system, that is, when the SINAP/SS7 system is ready to accept user registration parameters and send the UIS message. This script is executed only after all SINAP/SS7's processes have been started and are running. If you

use this script, make sure you modify it to include your installation's startup command(s).

- Include the client startup procedure in the UNIX initialization file (`/etc/inittab`) to respawn or start the client application process. With this method, the application process must call `ca_register()` until it is accepted (that is, until the function executes without returning an error).

  When its registration is accepted, the client application process must send the UIS message, using `ca_put_msg()`, until the message is accepted. When the SINAP/SS7 system accepts the registration and the UIS message, the client application process can communicate with the SINAP/SS7 system, accepting and sending network traffic.

- You can start the client application process using a script, or from the command line, after starting the SINAP/SS7 system. Whether you start the application process using a script or from the command line, you must follow the same rules as those for starting the application process through `/etc/inittab`.

Terminating a SINAP/SS7 Client Application

A client process can be terminated by means of the CASL interface, the UNIX operating system, or the failure of health-check operations.

- To terminate processing itself, an application process can call the CASL function `ca_terminate()`. An application's parent process can also call this function on behalf of its child processes. The `ca_terminate()` function contains a pointer to the *termination data packet*, which contains the parameters necessary for termination. The include file `$SINAP_HOME/Include/terminate.h` defines the format and content of the termination data packet.

  Optionally, you can use the `ca_withdraw()` function to have an application process terminate automatically once it completes its transaction. However, you must still call the `ca_terminate()` function to terminate the process.

- A process can terminate because of normal events, such as shutdown, or abnormal events, such as process corruption. The UNIX operating system can also terminate a client process for reasons such as a memory-protection violation or an operator-issued kill signal. To ensure that such situations are detected, the client application should use a parent process to spawn all operational processes. Then, the parent process will be notified when any operational processes are terminated by UNIX.

  NOTE

  This is the technique used by the SINAP/SS7 management processes.

- An application process can also be terminated due to the failure of a health-check operation (see the section Health-Check Operations earlier in this chapter).

TCAP Client Applications

A *TCAP client application* is a client application that interfaces with the SS7 network at the TCAP layer. Communication between two TCAP client applications typically involves one TCAP application requesting services from another (for example, a database look-up or a subscriber verification). TCAP client applications are the most common form of client applications. They consist of multiple transaction servers and management processes. It is assumed that the transaction servers are clones (re-entrantly-executed copies of a transaction server process).

Figure 3-2 shows a single management process controlling operational functions for the entire application and several cloned query/response processes. The management process interfaces to the SINAP/SS7 system at the SCCP protocol layer boundary and the cloned query/response server processes interface to the SINAP/SS7 system at the TCAP protocol layer boundary.

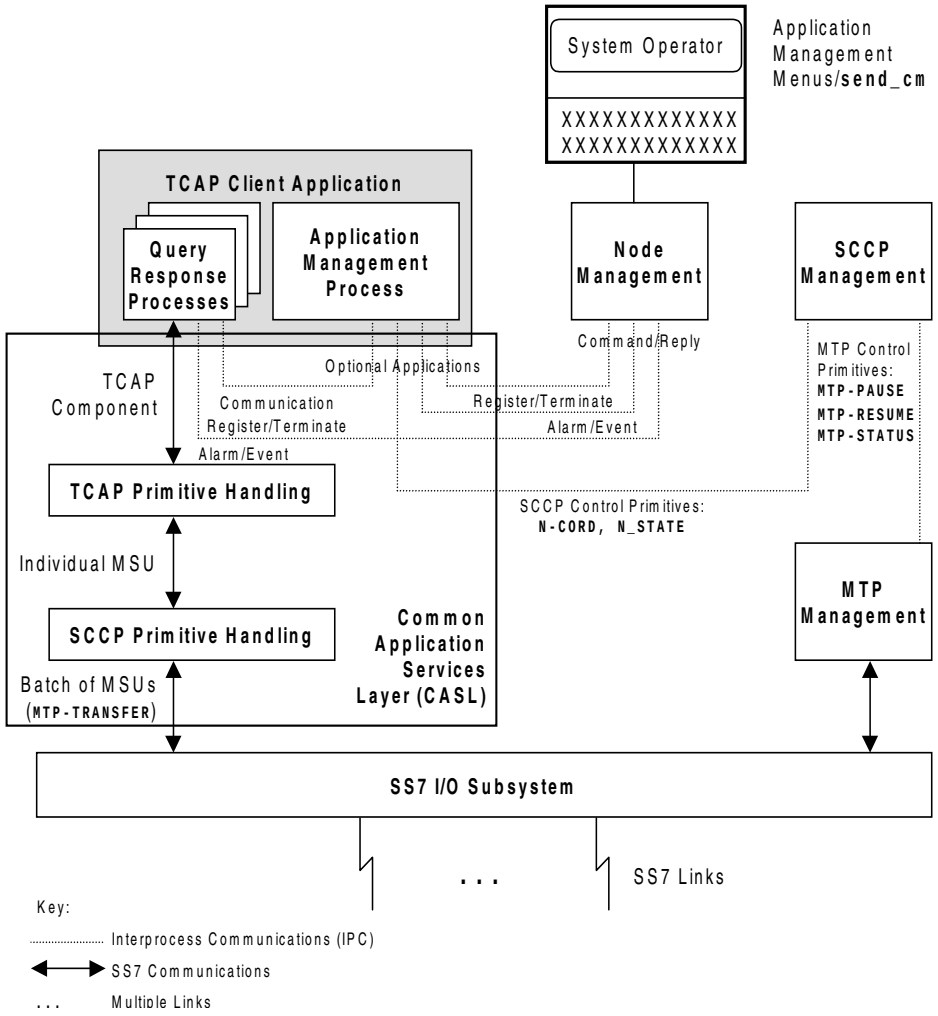

Figure 3-2. Typical TCAP Client Application

In a typical TCAP client application, the management process exchanges SCCP control primitives to manage its endpoints and its mate (if one exists). The cloned query/response processes are identical copies of one another. The number of these processes is determined by the application's design, performance requirements, and operational characteristics. For example, an application that can respond to a query or an incoming SS7 signaling event entirely from main memory-based resources uses a relatively small number of clone processes. Conversely, an application that performs extensive disk input/output (I/O) to respond to an incoming event uses many cloned query/response server processes to reduce the average

queuing delay for processing resources. The SINAP/SS7 system can support both types of applications by using a load distribution function that disperses incoming traffic across the cloned query/response server processes according to an application-selected algorithm.

The SINAP/SS7 system supports both the 1988 and 1993 editions of ITU-T (CCITT) Recommendations for TCAP (hereafter referred to as 1988 or 1993 TCAP standards). The 1988 TCAP standards are documented in the 1988 editions of the ITU-T Recommendations Q.771 through Q.775. The 1993 TCAP standards are documented in the 1993 editions of ITU-T Recommendations Q.771 through Q.775. You can develop TCAP applications that adhere to either set of standards. Existing TCAP applications that adhere to the 1988 standards will run without modification.

A node that implements the 1993 standards cannot interact with a node that implements the 1988 TCAP standards. However, many networks contain both kinds of nodes. This section describes how to implement both standards. Descriptions apply to both TCAP standards unless a specific standard is indicated. The section, Implementing 1993 TCAP Standards describes in detail how to implement those standards.

The 1988 TCAP standards are supported on all network variants. For the ANSI variant, TCAP supports the T1.114 Recommendations. The 1993 TCAP standards are only supported by the CCITT, China, NTT, and TTC network variants.

Communication Between TCAP Applications

When a local TCAP application wants to request services from another, the application initiates communication with the other application (the remote application). An *association* (a logical connection) is established between the applications using *application protocol data units* (APDUs) and *application service elements* (ASEs), described in this section.

Once communication is initiated, the applications can communicate until one or the other ends the session. Communication between two TCAP applications consists of all messages required to complete the requested operation: the local application's query or request and the remote application's response. A single communication might consist of multiple queries and responses.

NOTE

While designing your application, you should consider the types of applications with which your application will interface and the manner in which information will be communicated. As a developer, it is your responsibility to code your application so that it processes TCAP components in the appropriate manner.

The SINAP/SS7 system allows multiple TCAP applications to communicate simultaneously. To facilitate multiple communication sessions between one or more pairs of TCAP applications, the SINAP/SS7 system must identify each communication session. It does this by means of a

unique ID that is assigned for the duration of the communication session. This ID enables the TCAP to track multiple concurrent communication sessions between different TCAP applications.

- In the CCITT, China, NTT, and TTC variants of the SINAP/SS7 system, communication between two TCAP applications is called a *dialogue*. Each dialogue is identified by a unique *dialogue ID*. Based on ITU-T (CCITT) Recommendation Q.773, local dialogue IDs are exchanged between TCAP applications in the dialogue-handling portion of messages.
- In the ANSI variant of the SINAP/SS7 system, communication between two TCAP applications is called a *transaction*. Each transaction is identified by a unique *transaction ID*. Based on ANSI Recommendation T1.114.3, local transaction IDs are exchanged between TCAP applications in the transaction-handling portion of messages.

NOTE

To facilitate communication between the transaction sublayer (TSL) and the component sublayer (CSL), TCAP assigns its own unique ID to each dialogue/transaction; however, this ID is invisible to the TCAP application.

The methods for obtaining dialogue and transaction IDs differ. Each is described separately later in this chapter. The method for obtaining a dialogue ID is described in the section, Sending Outgoing Messages (CCITT/China/TTC/NTT) and the method for obtaining a transaction ID is described in the section, Sending Outgoing Messages (ANSI) later in this chapter."

Application Protocol Data Units (APDUs)

The association is established and supported by Application Protocol Data Units (APDUs) An ADPU is used to transmit information between two communicating applications. The 1988 and the 1993 TCAP standards use different types of APDUs.

- The 1988 standards use the following Remote Operations Service Element (ROSE) ADPUs:
  - `ROIV` (Invoke)
  - `RORS` (Result)
  - `RORJ` (Reject)
  - `ROER` (Error)
- In addition, the 1993 standards use the following association control service element (ACSE) APDUs:
  - AARQ (association request)
  - AARE (association response)
  - ABRT (association abort)
  - AUDT (unidirectional association request, which is the same as AARQ)

For detailed information about the ACSE ADPUs, see Section 4.2.3 of the 1993 edition of the ITU-T Recommendations Q.773.

An ASE identifies the particular communication protocol (such as TCP/IP, CMISE/CMIP, and ROSE) or the specific variant of a particular message protocol. Each ASE has a specification that defines its functionality and the message set it uses. An ASE is identified by an *object identifier* (OID) that defines the location of its specification. To be valid, an OID must be registered with the standards organization under which the ASE is defined. For example, an ASE defined by ITU-T (CCITT) standards must have an OID that is part of the ITU-T standards registration tree.

Sometimes two parties sign a joint implementation agreement (JOI), in which they agree to develop applications adhering to a particular predefined pattern for communication (a private ASE). In this case, the ASE's OID need not be registered with a standards organization, but both communication applications must adhere to the rules defined by that ASE.

An OID can take the form of an indirect reference or a direct reference to an ASE. The following two examples show both formats of a sample OID.

- Indirect reference:

```
dialogue-as-id OBJECT IDENTIFIER
             ::={ccitt recommendation q 773 as(1)
                 dialogue-as(1) version(1) }
```

- Direct reference:

```
dialogue-as-id OBJECT IDENTIFIER
             ::={ 0 17 134 5 1 1 1 }
```

In the direct reference, each number represents the corresponding element in the indirect reference. In the examples above, `0` represents `ccitt`, `17` represents `recommendation`, `134` represents `q`, and so on.

Maintaining Information about Transactions

The TCAP maintains information in the following two tables in order to manage multiple dialogue/transactions and to keep track of multiple TCAP components in a single dialogue/transaction.

- The dialogue/transaction ID table contains information about each active dialogue/transaction. When an application retrieves an ID for a dialogue/transaction, the SINAP/SS7 system provides the application with an entry from this table. Thereafter, the TCAP maintains information about the dialogue/transaction in that table entry.

  An association between each of the TCAP components in a single dialogue/transaction is formed by linking each component to the same entry in the dialogue/transaction ID table.

- The Invoke State Machine (ISM) table contains information about the state of each active operation invoked by a TCAP application. Each invocation of a particular operation is identified by an invoke ID, thus enabling several instances of a single operation to run concurrently. When TCAP forms a component from the `T_Blocks` that make up a single dialogue/transaction, the next available entry from the ISM table is linked to the transaction ID table entry for that dialogue/transaction.

TCAP Data Structure (`t_block_t`)

TCAP uses the `t_block_t` structure to communicate with the SINAP/SS7 system and with other TCAP applications. It contains all information required to initiate and maintain communication with the CASL and with the other TCAP application. For example, it contains the following types of information.

- Component-handling information for all network variants (`tc_chp_t` structure)
- Dialogue-handling information for CCITT, China, NTT, and TTC (`tc_dhp_t` structure)
- Transaction-handling information for ANSI (`tc_thp_t` structure)
- Addressing information (local process and remote process)
- Data and control information
- Error and problem codes

The 1993 TCAP standard supports a new field, `ahp`, in the `tc_dhp_t` structure to accommodate application-context information for the MSU. For more information, see the sections describing the 1993 TCAP standard.

Throughout this manual, the term `t_block_t` refers to all structures that collectively provide information about a TCAP component: `t_block_t`, `tc_chp_t`, and `tc_dhp_t` (CCITT, China, NTT, and TTC), or `tc_thp_t` (ANSI). Each structure is described in detail in "The T_Block Structure (t_block_t)" in Chapter 6 in the descriptions of the `ca_get_tc()` and `ca_put_tc()` functions.

Allocating `t_block_t` Structures

At registration, a TCAP application specifies the number of `t_block_t` structures that the SINAP/SS7 system should allocate for its use in the `register_req_t` structure's `tc_count` field. Based on the value of `tc_count`, the SINAP/SS7 system creates an array of `t_block_t` structures in the application's data space. This array is a joint work area for the client process and the TCAP functions within CASL. As the SINAP/SS7 system receives and decodes the MSUs, individual TCAP components are placed in the `t_block_t` structure. An index into the `t_block_t` array is returned in response to each primitive request from the client application. The client application can modify the specified `t_block_t` array entry and pass it back, or deallocate the entry to make it available again. All `t_block_t` structures are relative to the global pointer `PTB`, which is defined in the `ca_glob.h` include file. The client application can use the index to access the appropriate `t_block_t` structures by means of pointer arithmetic.

The client application process can also allocate one or more `t_block_t` structures entries for its own use. If the TCAP functions get these entries for output, the client application assumes that TCAP will deallocate them. If the client application keeps these entries, it must return them to the available pool when done

TCAP Application Include Files

When you develop a TCAP application, make sure the application references the following include files. Note that some of these include files, which are located in the `$SINAP_HOME/Include` directory, are **not** referenced by the `caslinc.h` include file.

- `caslinc.h` is the master CASL include file. It contains the include files most frequently used by SINAP/SS7 client applications.
- `tcap.h` defines the table and data structures used by the TCAP CSL and TSL.
- `scmg-prims.h` defines the structure formats of SCCP management primitives. Make sure the reference for `scmg-prims.h` follows the reference for `caslinc.h`. (This is because the file must be referenced after the `command.h` include file, which is referenced by `caslinc.h`.)
- `prims3.h` is required if the application will register to receive CONTROL primitives or CONTROL and DATA primitives. This include file defines the structures of MTP indication messages.

For 1993 TCAP standards, the `TC_DIALOGUE_REQUEST` structure is defined in the include file, `$SINAP_HOME_Include/tcglob.h`. (Note that `DIALOGUE_REQUEST` is the typedef for this structure. See `tcap.h`.) The structure temporarily stores the results of the APDU encoding or decoding. See APDU Encoding/Decoding Functions later in this chapter for a description of TCAP applications.

TCAP Application Registration

To register a TCAP client application process with the SINAP/SS7 system, code the application process so that it does at least the following tasks:

1. Initialize the global variable `CA_REG`, which is used by the registration process.
2. Initialize the `register_req_t` (`CA_REG`) structure to define the application's operating characteristics. For information about the operating characteristics that are specific to TCAP client applications, see the following section, "TCAP Registration Parameters."
3. Call the function `ca_register()` to register the application with node management. If there is a problem, `ca_register()` returns an error message. If this occurs, evaluate the error message and correct the problem.

   Before the application can receive and process SS7 data and/or control information, it must be activated and it must then go into service. For instructions on how to code your application to do this, see the section, "Going Into Service," earlier in this chapter.

TCAP Registration Parameters

As with all SINAP/SS7 client applications, a TCAP application must initialize the fields of the `register_req_t` structure to appropriate values before calling the `ca_register()` function. Specifically, the application must initialize the `register_req_t` structure's `sio_ssn_ind`, `sio_ssn`, and `ss7_input_boundary` fields to the values in Table 3-4. (You can specify either the symbolic value or the number in parenthesis.)

Table 3-4. `Register_req_t` **Structure Fields and Values**

Field	Value
`sio_ssn_ind`	`REG_SSN` (2)
`sio_ssn`	A decimal number in the range 2 through 255. This number must be unique among all SSNs of applications currently registered with the SINAP/SS7 system. All processes that are part of this application must register with the same SSN.
`ss7_input_boundary`	`SS7_INPUT_BOUNDARY_TCAP` (3)

NOTE

If the application implements the enhanced message distribution feature, initialize `sio_ssn_ind` to `REG_MULT` (3) and `sio_ssn` to `0`. (See Enhanced Message Distribution later in this chapter for additional instructions on implementing this feature.)

An application that consists of a control process and one or more data processes must observe the following rules when registering with the SINAP/SS7 system:

- The application's control process must register with a process name that is different than the process name used by the application's data processes. (The name of an application process is defined by the `register_req_t` structure's `proc` field.)
- Each application data process must register with the same process name, which is defined by the `register_req_t` structure's `proc` field.
- The application's control and data processes must each register with the same application name, which is defined in the `register_req_t` structure's `appl` field.

To define the types of primitives that the application process can receive, initialize the `register_req_t` structure's `fss7` and `ss7_primitive` fields as shown in Table 3-5:

Table 3-5. Primitive Fields

Process Type	`fss7`	`ss7_primitive`
Control	FALSE (0)	SS7_CTRL_PRIMITIVE (1)
Data	TRUE (1)	SS7_DATA_PRIMITIVE (2)
Control and Data	TRUE (1)	SS7_CTRL_DATA_PRIMITIVE (3)

Table 3-6 lists the types of primitives available to a TCAP application process. See TCAP Primitives in Chapter 2 for information about these primitives.

Table 3-6. TCAP Primitives

Primitive Type	Primitives Available
Data	TCAP components only
Control	I_N_COORD_REQ, I_N_COORD_INDIC, I_N_COORD_RESP, I_N_COORD_CONF, I_N_PCSTATE_INDIC, I_N_STATE_INDIC, I_N_STATE_REQ
Data and Control	I_N-PCSTATE, I_N_STATE_INDIC, I_N_STATE_REQ, I_N-COORD_REQ, I_N_COORD_INDIC, I_N_COORD_RESP, and I_N_COORD_CONF

In addition, the following `register_req_t` structure fields define additional TCAP operating characteristics that you might want to specify (see the description of `ca_register()` in Chapter 6 for more information):

- `tc_count` defines the number of `t_block_t` structures to allocate for the application.
- `max_dial_id` and `max_trans_id` define the number of dialogue IDs and transaction IDs, respectively, to allocate for the application.
- `max_ism` defines the maximum number of concurrent invocations allowed for the application.
- `tsl_timer_value` defines the amount of time the TSL has to process a message.

Handling Incoming SS7 Messages

When the SINAP/SS7 system receives an incoming MSU containing a TCAP component, TCAP dialogue- or transaction-handling primitives automatically activate the TCAP application to which the TCAP component is addressed and extract the TCAP component from the incoming MSU. The application is then responsible for retrieving and processing the TCAP component.

The procedure for processing a TCAP component differs slightly between the network variants of the SINAP/SS7 system. Each procedure is described separately in the sections that follow.

Processing Incoming Messages (CCITT/China/TTC/NTT)

For the CCITT, China, NTT, or TTC network variant, the procedure for processing an incoming MSU is basically the same for both the 1988 and 1993 TCAP standards. See the sections describing the differences in 1993 TCAP standards application, particularly "Processing an Incoming MSU" later in this chapter.

To process a TCAP component from another application, code your CCITT, China, NTT, or TTC TCAP client application so that it performs the following tasks:

1. Call the `ca_get_tc()` function to retrieve an incoming TCAP component.
2. Examine the TCAP component to determine how to process the incoming message.
   - If the incoming TCAP component contains a `TC_UNI` or `TC_BEGIN` primitive, `ca_get_tc()` retrieves the next available entry from the transaction ID table and assigns it to the dialogue (by specifying it in the `dialogue_id` field of the `t_block_t` structure).

     NOTE

     If the `ca_get_tc()` function is called with the `pfunc` parameter, `ca_get_tc()` calls the user-supplied function to which `pfunc` points. This user-supplied function assigns a TC-user transaction ID to the dialogue.

     Issue multiple calls to `ca_get_tc()` to retrieve all TCAP components in the incoming message.

   - If the incoming TCAP component contains a `TC_INVOKE` primitive, `ca_get_tc()` returns the index of the `T_Block` to the client application that initiated the dialogue.
   - If the incoming TCAP component contains a `TC_REJECT` or `TC_ABORT` primitive, `ca_get_tc()` calls an ISM function to process the request. It also generates the `REJECT` or `ABORT` component and returns an index of the `T_Block` to the client application that initiated the dialogue.

Processing Incoming Messages (ANSI)

To process a TCAP component from another application, code your ANSI TCAP client application so that it performs the following tasks:

1. Call the `ca_get_tc()` function to retrieve the next incoming TCAP component.
2. Examine the TCAP component to determine how to process the incoming message.
   - If the incoming TCAP component contains a `TC_UNI`, `TC_QRY_W_PERM`, or `TC_QRY_WO_PERM`, `ca_get_tc()` retrieves the next available entry from the transaction ID table and assigns it to the transaction (by specifying it in the `trans_id` field of the `T_Block`).

     NOTE

     If the `ca_get_tc()` function is called with the `pfunc` parameter, `ca_get_tc()` calls the user-supplied function to which `pfunc` points. This user-supplied function assigns a TC-user transaction ID to the transaction.

     Issue multiple calls to `ca_get_tc()` to retrieve all of the TCAP components in the incoming message.

   - If the incoming TCAP component contains a `TC_INVOKE_L` or `TC_INVOKE_NL` primitive, `ca_get_tc()` returns the index of the `T_Block` to the client application that initiated the transaction.
   - If the incoming TCAP component contains a `TC_REJECT` or `TC_ABORT` primitive, `ca_get_tc()` calls an ISM function to process the request. It also generates the `REJECT` or `ABORT` component and returns an index of the `T_Block` to the client application that initiated the transaction.

If the incoming message contains an SLS on the MTP routing label that the application needs to read, see "SINAP/SS7 Interaction with the SS7 Network" in Chapter 2 for information on using five-bit and eight-bit SLS processing schemes.

Sending Outgoing SS7 Messages

To send an outgoing message to a remote application, a SINAP/SS7 TCAP client application must initiate a dialogue/transaction, create one or more TCAP components for the message, and call the `ca_put_tc()` function to send the component(s) to the remote application. TCAP automatically packages the TCAP component(s) in an MSU and calls an internal CASL function to deliver the MSU to the SCCP. The SCCP then takes over processing and delivers the MSU to the MTP for transmission to its final destination.

The procedure sending an outgoing message differs slightly between the CCITT, China, TTC, NTT, and ANSI variants of the SINAP/SS7 system. Each procedure is described separately in the sections that follow.

Sending Outgoing Messages (CCITT/China/TTC/NTT)

To send a TCAP component to another application, include calls to the following CASL functions in your CCITT, China, TTC, or NTT TCAP application.

1. Call the function `ca_get_dial_id()` to obtain a unique ID for the dialogue. This function call returns the next available entry from the transaction ID table. If multiple dialogue IDs are required, call this function repeatedly.
2. Call the function `ca_alloc_tc()` to allocate a `T_Block` structure for the TCAP component. If the dialogue requires multiple TCAP components, call this function once for each component.
3. Create one or more TCAP components by initializing fields in the `t_block_t`, `tc_dhp_t`, and `tc_chp_t` structures. For more information about these structures, see The T_Block Structure (t_block_t) in the description of the `ca_put_tc()` function in Chapter 6.
   - Use the `priority` field in the `tc_dhp_t` field to specify the message priority for the MSU. This parameter is valid only for SCCP Class-0 and Class-1 messages.
   - Use the `seq_control` field in the `tc_dhp_t` structure to define the sequence control. This parameter is valid only for SCCP Class-1 messages.
   - Use the dialogue ID returned by `ca_get_dial_id()` or `ca_get_tc()` for the `t_block_t` structure's `dialogue_id` field. If the dialogue requires multiple TCAP components, use the same dialogue ID for each component. If multiple dialogue IDs are required, call this function repeatedly.
4. Call the function `ca_put_tc()` to send the TCAP component to its destination. If the dialogue requires multiple TCAP components, call this function once for each component.
5. End a normal dialogue by calling `ca_put_tc()` with the `t_block_t` structure's `primitive_code` field set to `TC_END` and the `tc_dhp_t` structure's `dialogue_end_type` field set to indicate how the dialogue is to end: `1` indicates a pre-arranged end; `2` indicates a basic end.

   For an application-context dialogue (1993 TCAP standard), you must use a basic end. Your application must call the `ca_put_tc()` function with the `t_block_t` structure's `primitive_code` field set to `TC_END`, and the `tc_dhp_t` structure's `dialogue_end_type` field set to `2`, which indicates a basic end.

   NOTE

   If `dialogue_end_type` is not `2`, the CASL returns an error.

6. Call the function `ca_dealloc_tc()` to deallocate the `T_Block` structure you allocated for the TCAP component. If the dialogue required multiple TCAP components, call this function once for each component.
7. Call the `ca_rel_dial_id()` function to release the dialogue ID and return it to the pool of available IDs. Otherwise, the dialogue ID remains unavailable until TCAP determines

that the dialogue has ended, in which case, TCAP automatically releases the dialogue ID.

Sending Outgoing Messages (ANSI)

To send a TCAP component to another application, include calls to the following CASL functions in your ANSI network variant TCAP application.

1. Obtain a unique transaction ID for the transaction by calling the `ca_get_trans_id()` function. TCAP assigns the ID. If multiple transaction IDs are required, call this function repeatedly.
2. Call the function `ca_alloc_tc()` to allocate a `T_Block` structure for the TCAP component. If the transaction requires multiple TCAP components, call this function once for each component.
3. Create one or more TCAP components by initializing fields in the `t_block_t`, `tc_thp_t`, and `tc_chp_t` structures. For more information about these structures, see The T_Block Structure (t_block_t) in the description of the `ca_put_tc()` function in Chapter 6.
   - Use the `priority` field in the `tc_thp_t` field to specify the message priority for the MSU. This parameter is only valid for SCCP Class-0 and Class-1 messages.
   - Use the `seq_ctrl` field in the `trans_id_t` structure to define a specific signaling line selection (SLS). This parameter is valid only for SCCP Class-1 messages.
   - In the `trans_id` field of the `t_block_t` structure, specify the transaction ID returned by the `ca_get_trans_id()`. If the transaction requires multiple TCAP components, use the same transaction ID for each component.
4. Call the function `ca_put_tc()` to send the TCAP component to its destination. If the dialogue requires multiple TCAP components, call this function once for each component.
5. End the transaction by calling `ca_put_tc()` with the `t_block_t` structure's `primitive_code` field set to `TC_RESPONSE` or `TC_NO_RESPONSE` and the `tc_dhp_t` structure's `trans_end_type` field set to indicate how the transaction is to end: `1` indicates a pre-arranged end; `2` indicates a basic end.
6. Call the function `ca_dealloc_tc()` to deallocate the `T_Block` structure you allocated for the TCAP component. If the transaction required multiple TCAP components, call this function once for each component.
7. Call the `ca_rel_trans_id()` function to release the transaction ID and return it to the pool of available IDs. Otherwise, the transaction ID remains unavailable until TCAP determines that the transaction has ended, in which case, TCAP automatically releases the transaction ID.

If the application sets a specific SLS in the MTP routing label or uses SCCP Class 1 service at the SCCP or TCAP levels and uses an eight-bit SLS, see "SINAP/SS7 Interaction with the SS7 Network" in Chapter 2 for more information.

1993 TCAP Standards Overview

In an advanced intelligent network (AIN), communication between applications is governed by a *message protocol*, which is a set of rules defining how messages are to be exchanged between applications. Every message protocol is registered with a standards organization and the rules defining that protocol are published in that organization's standards recommendations. For example, two Integrated Services Digital Network (ISDN) applications communicate by following the ISDN signaling standards defined in ITU-T Recommendation Q.763.

Typically, an application that provides services in an AIN (such as an application for translating 1-800 telephone numbers) is designed to operate with a particular message protocol. However, all the switches in the network might not support that protocol. To enable the application to support applications running on those switches, the 1993 TCAP standards allow an application to have one or more subapplications, each supporting a different variant of the application's message protocol.

To communicate with the application providing services in the AIN, an application running on a switch accesses the subapplication that supports its message protocol. The switch application identifies the subapplication by means of a dialogue portion included in the MSU. The dialogue portion contains an *application-context name* that identifies the ASE of the message protocol being used by the subapplication. The dialogue portion can also contain optional user information for the dialogue, such as a password or initialization information.

The subapplication handles information from the switch application and converts that information to the format required by the main AIN application, and vice versa. For example, an ISDN application accessed by applications running on three different types of switches might have three subapplications. Each subapplication is accessed by the particular switch application that uses that variant of the AIN application's message protocol.

When you specify the ASE of the subapplication with which you want your application to communicate, you are specifying the *application context* under which you want the dialogue to execute. Throughout this chapter, a dialogue that is initiated under a specific application context is referred to as an *application-context dialogue*.

Implementing 1993 TCAP Standards

The presence of a dialogue portion indicates that an MSU is part of an application-context dialogue. The *dialogue portion* contains an application-context name and optional user information. The *application-context name* specifies the application context to be used for the dialogue and the *user information* is any optional information that you want to use for the application-context dialogue (such as a password, application-initialization data, or protocol-version information).

The dialogue portion, which is placed between the transaction and component portions of an MSU, is defined in the CASL structures `tc_association_t`, `acn_t`, and `tc_user_data_t`. These structures are described in Chapter 6, ‘‘CASL Function Calls.”

The 1993 TCAP standards allow you to include a dialogue portion in the following types of messages.

- `TC_BEGIN`
- `TC_CONTINUE`
- `TC_END`
- `TC_U_ABORT`

To adhere to the 1993 TCAP standards, an application must contain the programming logic to process MSUs that contain a dialogue portion. An application that adheres to the 1988 TCAP standards need not contain this programming logic. (For examples of how to code your application to process MSUs that contain a dialogue portion, see the sample programs `dials.c` and `dialr.c` in the `$SINAP_HOME/Samples/ccitt` directory.)

Application-Context Names

At the start of a dialogue, the calling application specifies the name of the application context it wants to use by specifying the name of the ASE describing that context. This is the same ASE as is being used by the subapplication with which the calling application wants to communicate. If the called application supports that application context, both applications can continue communication using that application context. If not, the applications can negotiate for an application context they both support.

You must specify the application-context name as a properly formatted ASE OID, encoded according to the rules documented in ITU-T Recommendation X.690, *Basic Encoding Rules*. So that you do not have to code your application to encode the application-context name as an ASE OID, the CASL library provides the function, `tc_objmk()`, which automatically encodes the OID for you. For more information, see the section, “Defining an Application-Context Name,” later in this chapter.

Processing the Dialogue Portion of an MSU

The TCAP message-handling functions check incoming and outgoing MSUs for the presence of a dialogue portion. If the MSU contains a dialogue portion, the message-handling function automatically calls one of several APDU encoding/decoding functions to process the dialogue portion. These functions are described in the sections, TCAP Message-Handling Functions and APDU Encoding/Decoding Functions later in this chapter.

NOTE

The TCAP message-handling functions and the APDU encoding/decoding functions are both internal to the SINAP/SS7 system. You need not include them in your application.

TCAP Message-Handling Functions

The TCAP message-handling functions check incoming and outgoing MSUs for the presence of a dialogue portion. If one is present, the message-handling function hands off the processing of the dialogue portion to the appropriate APDU encoding/decoding function, based on the transaction's current state and the message type. For example, if the transaction is currently in an idle state and the application sends an outgoing `TC_BEGIN` message that contains a dialogue portion, the message-handling function `proc_req_A` calls the APDU encoding function `proc_req_A_dial` to create an APDU for the message. Likewise, upon receipt of an incoming MSU that contains a dialogue portion, the message-handling function calls the appropriate APDU decoding function to decode the MSU's dialogue portion and write the results to the MSU's application-context structures.

APDU Encoding/Decoding Functions

The dialogue portion for each type of message (`TC_BEGIN`, `TC_CONTINUE`, `TC_END`, and `TC_U_ABORT`) is associated with a particular type of APDU, each of which must be encoded or decoded according to the rules documented in ITU-T Recommendation X.690, *Basic Encoding Rules*. The SINAP/SS7 CASL library contains several internal APDU encoding/decoding functions that automatically perform the necessary encoding/decoding for you.

- *Encoding functions* create a properly encoded APDU for an outgoing MSU. These functions use the information in the application-context structures to create the APDU for the particular type of message defined in the MSU.
- *Decoding function*s extract the dialogue portion from an incoming MSU's APDU and write the results to the MSU's application-context structures.

The `TC_DIALOGUE_REQUEST` structure, which is defined in the include file `$SINAP_HOME/Include/tcglob.h`, is used to temporarily store the results of the APDU encoding or decoding function. When creating an outgoing MSU, the TCAP checks the size of the `TC_DIALOGUE_REQUEST` structure. If the structure's size is greater than 0, the structure's contents (the encoded dialogue portion) are copied into the MSU between the transaction and component portions; otherwise, the MSU is built without a dialogue portion.

Implementing 1993 TCAP Standards in Your Application

This section explains how to implement the 1993 TCAP standards in your SINAP/SS7 applications. The subsections discuss the following topics:

- Application-Programming Considerations highlights the major application programming considerations related to developing applications that implement the 1993 TCAP standards.
- Interaction Between Nodes provides information you will need if you are developing an application for use in a heterogeneous network consisting of nodes that adhere to the 1993 TCAP standards and nodes that adhere to the 1988 TCAP standards.
- Initiating an Application-Context Dialogue provides instructions for initiating an application-context dialogue. It also describes how to define the application-context name.

- Processing an Incoming MSU provides instructions for processing an incoming MSU that contains a dialogue portion.
- Ending an Application-Context Dialogue provides information about how to end an application-context dialogue.
- Error Handling for an Application-Context Dialogue describes conditions that cause the SINAP/SS7 system to abort an application-context dialogue.

Application-Programming Considerations

The following list highlights the major application programming considerations related to developing applications that implement the 1993 TCAP standards. For detailed information about these and other programming considerations, see the 1993 editions of ITU-T Recommendations Q.771 through Q.775.

- As you develop the code to implement the 1993 TCAP standards in an application, refer to the 1993 edition of ITU-T Recommendation Q.774, Figure A-5 (sheets 1 through 11). This figure illustrates the sequence of events occurring in an application-context dialogue. Although the SINAP/SS7 system executes many of the events automatically (for example, creating the appropriate APDU), the figure depicts the points at which various events occur and provides information on error handling.
- To specify the application-context name you want to use for a dialogue, the 1993 TCAP standards require that you use the direct-reference format of that application context's ASE OID. In addition, you must encode the OID according to the standards documented in ITU-T Recommendation X.690, *Basic Encoding Rules*.

  To have the SINAP/SS7 system automatically perform the necessary encoding, code your application to call the function `tc_objmk()`. For instructions, see the section, "Defining an Application-Context Name" later in this chapter.

  NOTE

  `objmk()` is a utility supplied with the ASN.1 compiler.

- Make sure your application contains logic to handle instances in which the called application proposes an application-context name that differs from the one your application proposed.
- Upon receipt of a `TC_BEGIN` message that contains an application-context name, your application must respond with a `TC_CONTINUE` message that contains the same application-context name.
- For incoming MSUs that are part of an application-context dialogue, your application need not examine the `tc_association_t` structure's `resultSourceDiag` and `resultSourceDiagValue` fields unless the `result` field is set to `rejected-permanent` (`1`).

- To end an application-context dialogue, the application should send a `TC_END` message that specifies a basic end. For instructions on how to code your application to do this, see the section, Ending an Application-Context Dialogue later in this chapter.

Interaction Between Nodes

A node that implements the 1993 TCAP standards (a *1993 TCAP node*) cannot interact with a node that implements the 1988 TCAP standards (a *1988 TCAP node*). However, many networks will probably continue to be made up of both types of nodes. Since your application might run in a heterogeneous network consisting of both types of nodes, you should consider the following issues when developing TCAP applications:

- A 1988 TCAP node cannot process an incoming MSU that contains a dialogue portion. If a 1988 TCAP node receives such an MSU, it will reject the MSU by sending the 1993 TCAP node an `ABORT` primitive whose reason code is set to `Incorrect Transaction Portion`.
- If your application sends an outgoing MSU that is rejected by the called application, you should determine why the MSU was rejected. You can do this by logging the error message and evaluating its contents.
  - If the MSU contains a syntax error, correct the error and resend the MSU.
  - If the MSU's syntax is correct, you can assume that the called application resides on a 1988 TCAP node and is therefore unable to process the MSU because it contains a dialogue portion. In this case, Stratus recommends you remove the dialogue portion and resend the MSU.

Initiating an Application-Context Dialogue

At the start of an application-context dialogue, the application issuing a request for services (the *calling application*) specifies an application-context name and optional user information. If the called application supports the specified application-context name, it accepts the request and continues with the application-context dialogue. Otherwise, it rejects the request or it proposes another application-context name. The calling application can then either continue with the dialogue using the new application context proposed by the called application, or it can issue an abort.

For detailed information about the sequence of steps executed by two applications engaged in an application-context dialogue, see the 1993 editions of ITU-T Recommendations Q.771 through Q.775 (specifically, Q.774, Figure A-5 (sheets 1 through 11)).

To initiate an application-context dialogue, you must code your application to perform the following steps, each of which is described in detail in the sections that follow.

- Define application-context information for the dialogue.
- Define an application-context name to use for the dialogue.
- Define optional user information for the dialogue.
- Initiate the dialogue.

Defining Application-Context Information

To define application-context information for the dialogue, make sure your application performs the following steps.

1. Initialize the `ahp` field of the `tc_dhp_t` structure to the name of the `tc_association_t` structure that contains application-context information for the dialogue. (The `tc_dhp_t` structure is part of the `t_block_t` structure.)
2. Initialize the fields of the `tc_association_t` structure as follows:
   - For `dlgInfoPresent`, specify the value `DIALOGUE_INDICATOR` or `FALSE(0)`.
   - For `applicationContextName`, specify the name of the `acn_t` structure that contains the application-context name to be used for the dialogue.
   - For `userInformation`, specify the name of the `tc_user_data_t` structure that contains any optional user information you want to include for the dialogue.

   For information about the `tc_association_t` structure's format, see the section, The tc_association_t Structure in Chapter 6.

Defining an Application-Context Name

To define an application-context name to use for the dialogue, make sure your application initializes the `acn_t` structure referenced in Step 2 of the section, "Defining Application-Context Information," earlier in this chapter. The `acn_t` structure specifies the direct-reference format of the application context's ASE OID, encoded according to the standards documented in ITU-T Recommendation X.690, *Basic Encoding Rules*. For information about the format of the `acn_t` structure, see "The acn_t Structure" in Chapter 6.

To have the SINAP/SS7 system automatically encode the ASE OID, your application should call the CASL function `tc_objmk()`, passing a string of numbers that represents the direct-reference format of the application context's ASE OID. The value returned by the `tc_objmk()` function is the properly encoded ASE OID for that application context. Make sure your application writes this value to the `acn_t` structure.

The `tc_objmk()` function accepts decimal or hexadecimal values. If you use hexadecimal notation, you must precede each hexadecimal value with the notation `0x`. (For example, the decimal notation, `0 17 134 5 1 1 1`, is `0x00 0x11 0x86 0x05 0x01 0x01 0x01` in hexadecimal notation.) Figure 3-3 shows a sample `tc_objmk()` function call. (Your `tc_objmk()` function call might be different, since the values you specify define the

direct-reference format of the ASE OID for the application context you are using.)

```
#include    <tcdialprocs.h>

    •
    •
    •

ptblk->tc_user.dhp.ahp.applicationContextName =
                  tc_objmk(0x00, 0x11, 0x86, 0x05, 0x01, 0x01,
                              0x01, END_OF_OID);
```

Figure 3-3. Sample `tc_objmk()` Function Call

To use the `tc_objmk()` function, your application must include one of the following statements. So that the application can take advantage of future enhancements, use the `#include` statement in your application. If you use the `extern` statement, place it at the end of the application's `#include` statements.

```
#include    <tcdialprocs.h>
```
or
```
extern acn_t    *tc_objmk();
```

NOTE

For C++ users, please refer to `tcdialprocs.h` for the C++ form of this `extern` statement as necessary.

NOTE

The `objmk()` function is supplied with the ASN.1 compiler.

Defining Optional User Information

To include optional user information in the dialogue portion, your application must initialize the `tc_user_data_t` structure referenced in Step 2 of the section," Defining Application-Context Information," earlier in this chapter. For information about the format of the `tc_user_data_t` structure, see "The tc_user_data_t Structure" in Chapter 6. If you do not plan to include optional user information in the dialogue portion, skip this section and proceed to the next section, "Initiating the Dialogue," in this chapter.

The `tc_user_data_t` structure contains the user information you want to use for the dialogue. You must format the structure's `userInfo` field according to Section 4.2.3 of the 1993 edition of ITU-T Recommendation Q.773. Table 49 of the recommendation indicates that user information must be defined as an ASN.1-encoded sequence of externals. You can define

a value for this field by using the ASN.1 compiler, which creates an ASN.1-encoded sequence of externals for you. You can then write the compilation results to the `userInfo` field.

Initiating the Dialogue

To initiate the dialogue, make sure your application performs the following steps:

1. Creates a TCAP component that contains a `TC_BEGIN` message by initializing a `t_block_t` structure and its associated structures. For information about the formats of these structures, see the description of the CASL function `ca_put_tc()` in Chapter 6, "CASL Function Calls."
2. Initiates the application-context dialogue by calling the CASL function, `ca_put_tc()`, to send the TCAP component you defined in the preceding step.

Processing an Incoming MSU

The procedure for processing an incoming MSU that is part of an application-context dialogue is basically the same as the procedure for processing a normal dialogue, which is described in the section, "Handling Incoming SS7 Messages," earlier in this chapter.

Upon receipt of an incoming MSU that contains a dialogue portion, the SINAP/SS7 system calls the appropriate APDU decoding function to decode the dialogue portion, writing the results to the `tc_association_t` structure and its associated structures, `acn_t` and `tc_user_data_t` (if user information is provided for the dialogue). The `tc_dhp_t` structure's `ahp` field points to the `tc_association_t` structure. Your application is responsible for retrieving the dialogue portion from these structures and processing both the MSU and its dialogue portion. For an example of how to code your application to perform this processing, see the sample program: `$SINAP_HOME/Samples/ccitt/dialr.c`.

You might want to code your application to examine the `dlgInfoPresent` field of the `tc_association_t` structure to determine whether the incoming MSU contains a dialogue portion. A value of `TRUE(1)` indicates the presence of a dialogue portion.

Ending an Application-Context Dialogue

Unlike a normal dialogue, which you decide how to end (by coding either a prearranged end or a basic end), you **must** end an application-context dialogue using a basic end.

To end an application-context dialogue, your application must call the `ca_put_tc()` function with the `t_block_t` structure's `primitive_code` field set to `TC_END` and the `tc_dhp_t` structure's `dialogue_end_type` field set to `2`, which indicates a basic end.

NOTE

If `dialogue_end_type` is not `2`, the CASL returns an error.

Error Handling for an Application-Context Dialogue

There are several ways your application can perform error handling. For detailed information about these error handling methods, as well as others, see the 1993 edition of ITU-T (CCITT) recommendations Q.771 through Q.775, specifically, Q.774, Figure A-5 (sheets 1 through 11).

- If your application does not support the application-context name proposed by another application, it must respond to the dialogue request with a `TC_U_ABORT` message indicating that the application-context name is not supported. To create the `TC_U_ABORT` message, your application must initialize the `t_block_t` structure's `primitive_code` field to `TC_U_ABORT` and it must initialize the `tc_association_t` structure's `resultSourceDiagValue` field to the value `application-context-name-not-supported`.
- When processing an incoming MSU, your application should check the `result` field of the `tc_association_t` structure to determine whether an error has occurred. A value of `1` indicates an error condition. If an error has occurred, have your application examine the `pduType` field of the MSU's `tc_association_t` structure.
  - **a.** If the field is set to `ABRT`, your application should examine the `abortSource` field of the `tc_association_t` structure to determine who aborted the dialogue. To obtain additional information about the abort, have the application examine any user information provided in the `ABORT` PDU.
  - **b.** If the field is set to `AARE`, your application should examine the `tc_association_t` structure's `resultSourceDiag` and `resultSourceDiagValue` fields to obtain information about the error condition that caused the abort. In addition, if an `ABORT` PDU was returned, have the application examine any user information it contains.

The SINAP/SS7 system automatically aborts an application-context dialogue when any of the following abnormal conditions occur.

- A `TC_U_ABORT` primitive is issued but the `ABORT REASON` parameter is missing, or it is set to a value other than `application-context-name-not-supported`.
- The SINAP/SS7 system receives an MSU containing an invalid dialogue portion.
- The SINAP/SS7 system aborts the underlying transaction with which the dialogue was associated.
- The TC user aborts the dialogue.

SCCP Client Applications

An SCCP client application is a variation of the TCAP client application. An SCCP client application has essentially the same organization as a TCAP client application. However, the transaction servers interface directly with the SCCP and do not use the services of the TCAP.

SCCP Application Include Files

When you develop an SCCP application, make sure the application references the following include files:

- `caslinc.h` is the master CASL include file. It contains the include files most frequently used by SINAP/SS7 client applications.
- `prisms3.h` is required if the application will register to receive CONTROL primitives or CONTROL and DATA primitives. This include file defines the structures of MTP indication messages. Make sure the reference to `prims3.h` follows the reference for `caslinc.h`.

> NOTE
>
> These include files are located in the `$SINAP_HOME/Include` directory.

The following two include files are required if the application handles SS7 traffic (that is, accepts incoming MSUs and/or processes responses in outgoing MSUs).

- `sccphdrs.h` defines the general SCCP header and the headers for SCCP connection requests.
- `sccp-intrn.h` is the SCCP internal header file.

If the application uses the connection-oriented feature (COF) for class 2 and 3 services, it must include the following files:

- `scoc-prims.h` defines the format for the SCCP connection-oriented control primitives.
- `sc23.h` provides prototypes and general COF defines.

SCCP Application Registration

To register an SCCP client application process with the SINAP/SS7 system, code the application process so that it does the following:

1. Initialize the global variable `CA_REG`, which is used by the registration process.
2. Initialize the `register_req_t` (`CA_REG`) structure to define the application's operating characteristics. For information about the operating characteristics that are specific to SCCP client applications, see the following section, "SCCP Registration Parameters."
3. Call the function `ca_register()` to register the application with node management. If there is a problem, `ca_register()` returns an error message. If this occurs, evaluate the error message and correct the problem.

   Before the application can receive and process SS7 data and/or control information, it must be activated and it must then go into service. For instructions on how to code your application to do this, see the section, "Going Into Service," earlier in this chapter.

SCCP Registration Parameters

As with all SINAP/SS7 client applications, an SCCP application must initialize the fields of the `register_req_t` (`CA_REG`) structure to appropriate values before calling the `ca_register()` function. Specifically, the application must initialize the `register_req_t` structure's `sio_ssn_ind`, `sio_ssn`, and `ss7_input_boundary` fields to the values in Table 3-7. You can specify either the textual value or the number in parenthesis.

Table 3-7. `register_req_t` Structure Parameters and Values

Parameter	Value
`sio_ssn_ind`	`REG_SSN` (2)
`sio_ssn`	A decimal number in the range 2 through 255. This number must be unique among all of the SSNs of applications currently registered with the SINAP/SS7 system. All processes that are part of this application must register with the same SSN.
`ss7_input_boundary`	`SS7_INPUT_BOUNDARY_SCCP (2)`

NOTES

1. If the application implements the enhanced message distribution feature, initialize `sio_ssn_ind` to `REG_MULT` (3) and `sio_ssn` to `0`. Then see Enhanced Message Distribution later in this chapter for additional instructions on implementing this feature.
2. If the application implements the custom application distribution feature, it will appear to be identical to enhanced message distribution for the purposes of SCCP management. see Custom Application Distribution later in this chapter for additional instructions on implementing this feature.

An application that consists of a control process and one or more data processes must observe the following rules when registering with the SINAP/SS7 system.

- The application's control process must register with a process name that is different than the process name used by the application's data processes. (The name of an application process is defined by the `register_req_t` structure's `proc` field.)
- Each of the application's data processes must register with the same process name, which is defined by the `register_req_t` structure's `proc` field.
- The application's control and data processes must each register with the same application name, which is defined in the `register_req_t` structure's `appl` field.

To define the types of primitives that the application process can receive, initialize the `register_req_t` structure's `fss7` and `ss7_primitive` fields as shown in Table 3-8.

Table 3-8. Primitive Types

Process Type	`fss7`	`ss7_primitive`
Control	`FALSE (0)`	`SS7_CTRL_PRIMITIVE (1)`
Data	`TRUE (1)`	`SS7_DATA_PRIMITIVE (2)`
Control and Data	`TRUE (1)`	`SS7_CTRL_DATA_PRIMITIVE (3)`

Table 3-9 lists the types of primitives available to an SCCP application process. See "SCCP Primitives" in Chapter 2 for information about these primitives.

Table 3-9. Primitives Available to SCCP Applications

Primitive Type	Primitives Available
Data	TCAP components only
Control	`I_N_COORD_REQ, I_N_COORD_INDIC, I_N_COORD_RESP, I_N_COORD_CONF, I_N_PCSTATE_INDIC, I_N_STATE_INDIC, I_N_STATE_REQ`
Data and Control	`I_N-PCSTATE, I_N_STATE_INDIC, I_N_STATE_REQ, I_N-COORD_REQ, I_N_COORD_INDIC, I_N_COORD_RESP, and I_N_COORD_CONFIG`

SCCP Application Message Processing

An SCCP application uses the `ca_put_msu()` and `ca_get_msu()` functions to transfer MSUs to and from the SS7 network.

- To retrieve an incoming MSU addressed to it, the application issues a call to the `ca_get_msu()` function.
- To send an outgoing MSU to the network, the application issues a call to the `ca_put_msu()` function, specifying the remote SCCP user with which it wants to communicate.

For the ANSI network, if an SCCP application needs to set or get an eight-bit SLS in the MTP routing label, see "SINAP/SS7 Interaction with the SS7 Network" in Chapter 2 for more information.

User Part (MTP) Client Applications

A user part client application uses the services of the MTP only; it does not use the services of TCAP or SCCP. Figure 3-4 shows a typical user-part client application. In this figure, assume that load distribution is disabled and that an inbound MSU performs the distribution function. Though the SINAP/SS7 system allows load distribution to be bypassed, the facility is available to MTP users.

The sample user part client application depicted in Figure 3-3 would be appropriate for a TUP application.

NOTE

The SINAP/SS7 system provides an ISUP layer currently. However, in the past, ISUP applications have been developed as MTP client applications.

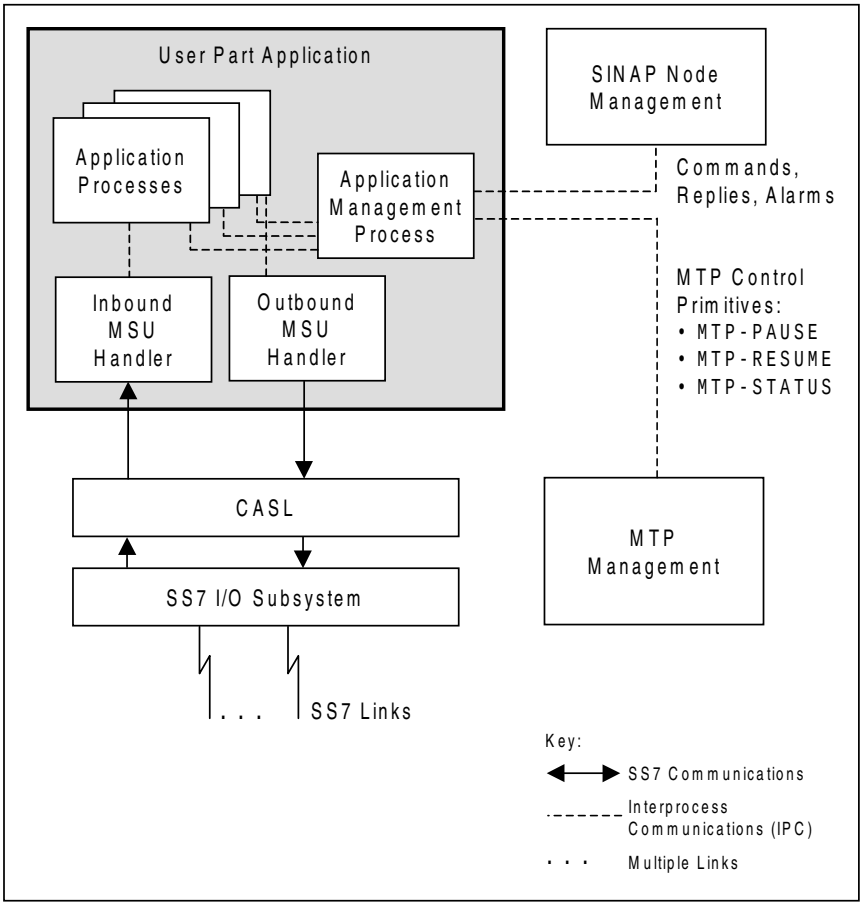

Figure 3-4. Typical User Part Client Application

When the application acts as the user part, the inbound SS7 function uses the SIO to route inbound `MTP-TRANSFER` primitives to it. MTP Management provides MTP control primitives (`MTP-PAUSE`, `MTP-RESUME`, and `MTP-STATUS`) using IPC.

User Part (MTP) Application Include Files

When you develop an MTP application, make sure the application references the following include files.

- `caslinc.h` is the master CASL include file. It contains the include files most frequently used by SINAP/SS7 client applications.

- `prims3.h` is required if the application will register to receive CONTROL primitives or CONTROL and DATA primitives. This include file defines the structures of MTP indication messages.

User Part (MTP) Application Registration

To register an MTP client application process with the SINAP/SS7 system, code the application process so that it does the following:

1. Initialize the global variable `CA_REG`, which is used by the registration process.
2. Initialize the `register_req_t` (`CA_REG`) structure to define the application's operating characteristics. For information about the operating characteristics that are specific to MTP client applications, see the User Part (MTP) Registration Parameters section which follows.
3. Call the function `ca_register()` to register the application with the SINAP/SS7 Node Management. If there is a problem, `ca_register()` returns an error message. If this occurs, evaluate the error message and correct the problem.

   Before the application can receive and process SS7 data and/or control information, it must be activated and it must then go into service. (For instructions on how to code your application to do this, see Going Into Service earlier in this chapter.)

User Part (MTP) Registration Parameters

As with all SINAP/SS7 client applications, an MTP client application must initialize the fields of the `register_req_t` structure to appropriate values before calling the `ca_register()` function. Specifically, the application must initialize the `register_req_t` structure's `sio_ssn_ind`, `sio_ssn`, and `ss7_input_boundary` fields to the values in Table 3-10. (You can specify either the symbolic value or the number in parenthesis.)

Table 3-10. `register_req_t` Structure Parameters and Values for MTP

Parameter	Value
`sio_ssn_ind`	`REG_SIO` (`1`)
`sio_ssn`	A decimal number in the range `1` through `15`. This number must be unique among all of the SIOs of applications currently registered with the SINAP/SS7 system. All processes that are part of this application must register with the same SIO.
`ss7_input_boundary`	`SS7_INPUT_BOUNDARY_MTP` (`1`)

An application that consists of a control process and one or more data processes must observe the following rules when registering with the SINAP/SS7 system.

- The application's control process must register with a process name that is different than the process name used by the application's data processes. (The name of an application process is defined by the `register_req_t` structure's `proc` field.)
- Each of the application's data processes must register with the same process name, which is defined by the `register_req_t` structure's `proc` field.
- The application's control and data processes must each register with the same application name, which is defined in the `register_req_t` structure's `appl` field.

To define the types of primitives that the application process can receive, initialize the `register_req_t` structure's `fss7` and `ss7_primitive` fields as shown in Table 3-11:

Table 3-11. Primitive Types Received for MTP Application

Process Type	`fss7`	`ss7_primitive`
Control	`FALSE (0)`	`SS7_CTRL_PRIMITIVE` (1)
Data	`TRUE (1)`	`SS7_DATA_PRIMITIVE` (2)
Control and Data	`TRUE (1)`	`SS7_CTRL_DATA_PRIMITIVE` (3)

Table 3-12 lists the types of primitives available to an MTP application process. See MTP Primitives in Chapter 2 for information about these primitives.

Table 3-12. Primitives Available to MTP Applications

Primitive Type	Primitives Available
Control	`MTP-PAUSE, MTP-RESUME, MTP-STATUS`
Data and Control	`MTP-PAUSE, MTP-RESUME, MTP-STATUS, MTP-TRANSFER`

User Part (MTP) Application Message Processing

An MTP application uses the `ca_put_msu()` and `ca_get_msu()` functions to transfer MSUs to and from the SS7 network.

- To retrieve an incoming MSU, the application issues a call to the `ca_get_msu()` function.
- To send an outgoing MSU to the network, the application issues a call to the `ca_put_msu()` function.

For the ANSI network, if an MTP application needs to set or get an eight-bit SLS in the MTP routing label, see "SINAP/SS7 Interaction with the SS7 Network" in Chapter 2 for more information.

MTP Routing Based on SLS and DPC

In the CCITT or China network variants of the SINAP/SS7 system, MTP-boundary users can ensure sequentiality of messages by routing based solely on the signaling link selection (SLS) and destination point code (DPC). For example, if a telephone user part (TUP) user employs the CIC for the SLS value for all messages pertaining to that circuit, all messages will go out over the same link to a particular DPC, even when load sharing over two link sets, thus ensuring they all remain in sequence. However, for any one DPC, only 8 links in each link set can be used when load sharing over link sets. With multiple DPCs and more than 8 links per link set, an even load distribution might be achieved, but there is no guarantee. This feature only pertains to the CCITT and China variants, which use a 4-bit SLS (16 possible values).

To enable MTP routing based on SLS and DPC, define the following environment variable before starting or restarting the SINAP/SS7 system:

```
MTP_SLS4_LOAD_SHARE
```

You can define the variable by uncommenting it in your `$SINAP_HOME/Bin/sinap_env.[sh or csh]` file to have the variable defined automatically each time you start the SINAP/SS7 system.

If the `MTP_SLS4_LOAD_SHARE` environment variable is not set, the default action for CCITT and CHINA variants is to support 16 links per link set, with a random choice of linkset when load sharing. This ensures even distribution, but not sequentiality.

NOTES

1. This option does not affect the TCAP or SCCP user. It also does not affect the ISUP user, as link set selection and link selection over a possible 16 links per link set is done transparently to the user by the SINAP ISUP code.
2. If you are running the MultiStack product, you must set the environment variable separately for each node where you want to activate this feature.

In the ANSI network, MTP-boundary users can ensure sequentiality of messages by setting either a five-bit or eight-bit SLS value in the MTP routing label (see "SINAP/SS7 Interaction with the SS7 Network" in Chapter 2 for more details). If the system user selects an eight-bit SLS (using the `CHANGE-SLSTYPE` MML command) and the application sets an eight-bit SLS value, the SINAP node maps the value to the signaling link code (SLC) for a specific link, which remains the same until a network event such as a changeover/changeback occurs. If the system user selects a five-bit SLS and the application sets an eight-bit SLS value, the SINAP node

masks out the upper three most significant bits of the SLS. However, the truncated SLS still maps to a specific link. To see how the SLS maps to a specific link, use the `sy`, `#ort,ls` or the `#ort,cls` command (if using a combined link set) to index the SINAP SLC.

ISUP Services Applications

The *ISUP services* feature supports the services provided by the ISDN User Part (ISUP) protocol of Signaling System Number 7 (SS7). ISUP provides the signaling functions required to support circuit-switched services for voice and non-voice connections to an integrated services digital network.

By default, the ISUP services feature is turned off. To activate the ISUP services feature, you must define an environment variable before starting or restarting the SINAP/SS7 system. For detailed information on implementing ISUP services applications, see the *SINAP/SS7 ISDN User Part (ISUP) Guide* (R8053).

Considerations for Implementing SINAP/SS7 Features

This section describes things you should be aware if you plan to implement any SINAP/SS7 features in an application. For considerations on implementing ISUP services features, see the *SINAP/SS7 ISDN User Part (ISUP) Guide* (R8053). This section discusses the following topics.

- *Routing capabilities* enable all variants of the SINAP/SS7 system to perform SCCP routing based on a global title address or to perform global title translation. The fictitious originating point code (FOPC) feature, available only in the ANSI variant, enables the SINAP/SS7 system to specify an OPC in the MTP routing label that is different than the SCCP calling party address's point code of the outbound MSU.
- *Enhanced message distribution* enables a SINAP/SS7 application to register with multiple SSNs or to register to receive input from specific OPCs. This feature is available only to applications that interface with the SINAP/SS7 system at the SCCP or TCAP boundary.
- *SCCP Third Party Address* field facilitates routing between an SSP/node, a TCP/IP agent, and an application.
- *Custom Application Distribution (CAD)* enables TCAP applications to register for application distribution based on the ServiceKey message parameter of the ETSI CS1 INAP initialDP invoke operation. This feature extends the capabilities provided by the enhanced message distribution feature.
- *Multiple link congestion levels* provide the following ITU-T (CCITT) congestion options for the CCITT and China variants of the SINAP/SS7 system: National Multiple Congestion States with Congestion Priority option (Q.704, 3.8.2.2 and National Multiple Congestion States without Congestion Priority option (Q.704, 3.8.2.3) *in addition* to the default option: International One Congestion Onset and One Congestion Abatement option (Q.704, 3.8.2.1).

- *MTP User Flow Control* enables the SINAP/SS7 system to generate a User Part Unavailable (UPU) message when the user part is unavailable (for example, the ISDN User Part).
- *Extended unitdata (XUDT)* and *extended unitdata service (XUDTS)* messages can be exchanged by applications running on networks configured for CCITT or China.
- The CCITT network variant can be configured to enable a SINAP node to receive and process *SCCP subsystem tests in an XUDT message*. You activate this feature by setting an environment variable.
- *SNM messages with nonzero SLCs* enables the SINAP/SS7 system to use a nonzero SLC value for any SNM message that is not related to a signaling link. This feature is available for the CCITT, China, and ANSI network variants.
- The *MTP restart* process helps prevent routing problems that can occur after the system resumes sending user traffic following a failure due to invalid routing information or too many parallel activities.
- *MTP time-controlled changeover* supports handling of long-term or short-term processor outages or changeover orders received from the remote end during the MTP Level 3 T1 timer period.
- *MTP time-controlled diversion* delays changeback to avoid missequencing messages to destination points after a remote point code restarts. This feature is only available for the ANSI network variant.
- *MTP Management Inhibit* supports management inhibit procedures for link-forced uninhibit request messages based on 1992 ANSI standards and 1988 ANSI standards for MTP.
- *Priority, sequence control, and quality of service parameters* enable you to specify how the SINAP/SS7 system should process outgoing MSUs.
- *SLS Message Distribution* enables the SINAP/SS7 system to distribute an incoming MSU based on the value of its SLS field. This feature is available only to applications that interface with the SINAP/SS7 system at the MTP or SCCP boundary.
- Eight-bit or five-bit SLS processing scheme selection enables use of eight-bit or five-bit SLSs for all incoming and outgoing messages. This feature is available in the ANSI network variant only.
- *Connection-oriented services* enable an application to establish and maintain a connection or logical communication path with another application for the purpose of exchanging small and large messages.
- *Load control* reduces the risk that incoming MSUs will be lost or discarded during times of severe network congestion. This feature is available only to applications that interface with the SINAP/SS7 system at the TCAP boundary.
- *Loopback detection* enables the SINAP/SS7 system to detect when a remote link is in loopback mode. This feature is available for the CCITT network variant only.

- *Transfer-restricted message handling* enables the CCITT network variant to use the national network option for restricting message traffic routing.
- The ANSI network variant can be configured to issue an RSR/RSP in response to TFR/TFP messages either *before* or *after* an MTP Level 3 T10 timer expires. Set an environment variable to select either network behavior option. These options conform to either the 1988 or the 1992 ANSI Standards behavior.

Routing Capabilities

This section describes the following routing capabilities available for Transaction Capabilities Application Part (TCAP) messages:

- Global Title Addressing (GTA)
- Global Title Translation (GTT)
- Fictitious originating point code (FOPC) feature
- Alternative Destination Point Code (ADPC) feature

Global Title Addressing (GTA)

The SINAP/SS7 system supports SCCP routing based on a global title, which enables SINAP/SS7 application processes to access the global title element of an SCCP called- or calling-party address. In global title addressing, the SINAP/SS7 system does not perform global title address translation; the application must perform the translation.

NOTE

The SINAP/SS7 system allows both hexadecimal and numeric values (A-F and 0-9) to be used in Global Title strings when the environment variable `HEX_GLOBAL_TITLE` is defined.

To use the global title addressing feature, define the environment variable `BYPASS_SINAP_GLOBAL_TITLE_TRANSLATION` before starting the SINAP/SS7 system. If you do not define this variable, GTT is implemented, as described in the next section.

The ability to route on global titles affects SINAP/SS7 functionality in the following ways:

- An outgoing TCAP message can specify a called party address that contains both a global title and an SSN of 0. The message is sent regardless of the status of the destination.
- An incoming TCAP message, whose called party address contains a global title, is passed to the application. The SINAP/SS7 system performs syntax checking on the global title and rejects the message if the syntax is incorrect.

Global Title Translation (GTT)

A global title is a type of address, such as a series of dialed digits, that does not itself contain the addressing information necessary to route a message signaling unit (MSU) to its destination.

Instead, SCCP must perform *global title translation* (GTT), a process that translates the global title into addressing information that can then be used to route a message to its destination.

> N O T E
>
> To bypass GTT in the SINAP/SS7 SCCP layer and implement global title addressing capabilities instead, define the environment variable, `BYPASS_SINAP_GLOBAL_TITLE_TRANSLATION` before starting the SINAP/SS7 software. This allows you to pass messages containing a global title without actually translating the global title.

For SCCP to perform GTT, you must provide information about how you want each global title translated. You do this by creating global title entries using the `CREATE-GTT` command. Each entry describes a particular global title and specifies how it should be translated. You can define a maximum of 4000 global title entries. You can specify a global title to be translated into a DPC, SSN, a new global title, or any combination thereof. The global title entry is stored in a segment of shared memory known as the *GTT table*.

> N O T E
>
> The GTT table is not a table in the true sense of the word; it is simply a storage area for global title entries.

SCCP performs GTT when the address indicator of the SCCP's **called party address** of an inbound or outbound message is set to indicate routing on global titles.

Address Indicator

The first octet (that is, eight bits) of the SCCP called- or calling-party address contains an address indicator that specifies whether the address contains a DPC, an SSN, and/or a global title. SCCP determines how to route an MSU from the information in the SCCP **called party**

address. Figure 3-5 shows the address indicator's format, which differs slightly between variants.

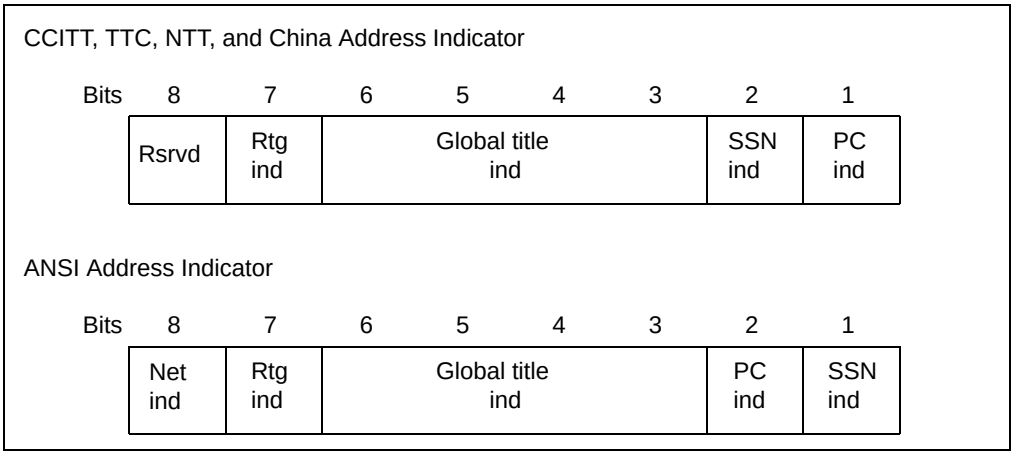

Figure 3-5. Address Indicator Formats

Descriptions of the bits in the address indicator follow:

- Bit 8 is not used in the CCITT, TTC, NTT, or China network variants. It is reserved for future use. In the ANSI variant, this bit signifies the network type: 0 for international and 1 for national.
- Bit 7, the routing indicator, indicates how SCCP should route the MSU. A value of 0 indicates routing on global title, and a value of 1 indicates routing on DPC and SSN.
- Bits 6 through 3, the global title indicator, define the address components included in the global title. For more information about global title indicators, see Chapters 3 and 4 of the *SINAP/SS7 User's Guide* (R8051).
- In CCITT, TTC, NTT, and China, bits 2 and 1 have the following meaning.
  - Bit 2, the SSN indicator bit, is set to 1 to indicate that the address contains an SSN.
  - Bit 1, the point-code (PC) indicator, is set to 1 to indicate that the address contains a DPC or an origination point code (OPC).
- In ANSI, bits 2 and 1 have the following meaning.
  - Bit 2, the PC indicator, is set to 1 to indicate that the address contains a DPC or an OPC.
  - Bit 1, the SSN indicator bit, is set to 1 to indicate that the address contains an SSN.

Global Title Format

A global title consists of several different types of address components, each defining a particular characteristic of the global title. The global title format, as defined by the global title indicator (GTI), determines the address components that are included in a global title.

Table 3-13 describes the address components that make up a global title. See the appropriate documentation for your network variant for more information on GTT components.

Table 3-13. Global Title Address Components

Address Component	Description
Global Title Indicator (GTI)	Indicates the global title format. The GTI value specifies which address components are included in the global title.
Translation Type (TT)	Directs the MSU to the appropriate global title translation function.
Numbering Plan (NP)	Indicates the numbering plan used for the global title's address information (for example, ISDN/Telephony, ISDN/Mobile, or data).
Encoding Scheme (ES)	The part of the numbering plan that indicates the type of encoding, if any, that is used. The SINAP/SS7 system uses the ES to determine whether the global title address information is an ODD (ES=1) or EVEN (ES=2) number of BCD digits.
Nature-of-Address Indicator (NOAI)	Indicates the type of number used in the global title (for example, a subscriber number, a national significant number, or an international number). (Not used for the ANSI network variant.) **Note:** The default valid range for this indicator is 1 through 4. If the `GTT_BYPASS_NOAI_CHECK` environment variable is defined for a node, the valid range is from 0 through 127. The variable must be defined for each node in a SINAP/SS7 environment. You can uncomment the environment variable in the `sinap_env.sh` or `sinap_env.csh` file where it will be automatically defined each time you start the SINAP node.

Table 3-14 lists the global title format associated with each GTI value. Note that the global title formats differ slightly between the network variants.

Table 3-14. GTI Values and Global Title Formats

CCITT, China, TTC, and NTT Global Title Formats		ANSI Global Title Formats	
GTI Value	**Global Title Contents**	**GTI Value**	**Global Title Contents**
0	No global title	0	No global title
1	NOAI only	1	TT, NP, and ES†
2	TT only	2	TT only
3	TT, NP, and ES		
4	TT, NP, ES, and NOAI		

† The global title formats for ANSI GTI 1 and CCITT/China/TTC/NTT GTI 3 are identical.

GTT Processing

The `CREATE-GTT` command creates a global title entry describing the global title and how it should be translated.

Table 3-15 describes how the CREATE-GTT command arguments correspond to the global title address components.

Table 3-15. `CREATE-GTT` Arguments

Arguments	Description
`GTI, NOAI, TT, NP`	Defines the address components included in the global title, as defined by the `GTI`. For example, a global title with a `GTI` of 1 contains the `NOAI` and `LADDR` address components. A global title with a `GTI` of 4 contains the `TT`, `NP`, `NOAI`, and `LADDR` address components.
`LADDR, HADDR`	Defines the global title's address information
`DPC, SSN, NADDR`	Defines the new `DPC`, `SSN`, and/or address information that replaces the original global title
`DPC2, SSN2`	Define the new DPC and SSN information if the original SCCP is unavailable.

When performing SCCP routing control for both inbound and outbound MSUs, the SINAP/SS7 system examines the **called party address** field in the SCCP header. If the address indicator

specifies routing on global title, the SINAP/SS7 system searches the global title entries for one that matches the global title in the called address. A match occurs when all the global title's address components are the same as the values in a particular global title entry. To compare the address information in a global title to the address information in a global title entry, both sets of address information must be the same length. For example, the 9-digit string, 180055512, would not match global titles in the range, 1800555111 to 1800555444 (which are each 10 digits long).

If the SINAP/SS7 system finds a global title entry with the same GTI, TT, NP and/or NOAI as the global title in the SCCP called party address, SCCP then examines the low address (LADDR) and high address (HADDR) of the matching global title entry. If only the LADDR is specified, SCCP looks for an exact match of the LADDR and the global title. If the LADDR and the HADDR are both specified, SCCP looks to see if the global title is lexically greater than or equal to the LADDR and less than or equal to the HADDR.

If the global title match is successful, SCCP replaces the DPC, SSN, and/or address information in the SCCP called party address with the replacement values defined in the global title entry. If SCCP replaces the global title's address information, the SINAP/SS7 system resets the SCCP called party address routing indicator bit to 0 to specify routing on global title. If SCCP does not replace the global title's address information, the SINAP/SS7 system sets the SCCP called party address routing indicator bit to 1 to specify routing on DPC and SSN.

The translation of a global title yields one of the following results.

- DPC
- SSN
- GT
- Any combination of DPC, SSN, and GT
- For CCITT only, DPC2 and SSN2 for alternate routing if the primary SCCP is unavailable (see Alternate SCCP Routing later in this chapter for more information)
- The translation for the GT does not exist

If the translation of a global title in an incoming MSU results in a DPC that differs from your node's own point code, SCCP attempts to reroute the MSU to the new DPC.

Alternate SCCP Routing

The CCITT network variant has the optional ability to route global title-related MSUs to an alternate or "backup" SCCP if the "primary" one is unavailable.

Two types of GTTs can result in MSUs being sent to the backup DPC and/or SSN. The two GTT types are:

- `RouteOnGT`
- `RouteOnSSN`

RouteOnGT translates the global title to another global title that requires further GTT at a remote DPC's SCCP. For RouteOnGT, the status of any remote subsystems would usually be immaterial.

To prevent testing the availability of the remote subsystems, set the environment variable GLOBAL_TITLE_SSN_NO_CHECK. To make this environment variable permanent, uncomment it in the $SINAP_HOME/Bin/sinap_env.[sh or csh] before starting the SINAP/SS7 system:

```
GLOBAL_TITLE_SSN_NO_CHECK=1
```

RouteOnSSN locally translates the global title into a remote DPC and SSN. For RouteOnSSN, the availability status of the remote subsystem is important and the MML should create the appropriate remote subsystem entries.

During normal system operation, the SINAP/SS7 system translates the global title either to another global title for translation at a remote primary DPC (RouteOnGT) or to DPC/SSN for processing at a remote primary DPC (RouteOnSSN). If the primary is not operational, the SINAP/SS7 system attempts to route the information to the backup DPC/SSN (DPC2/SSN2).

To specify an alternate SCCP for GTT, specify the DPC2 and/or the SSN2 parameters. When the DPC parameter is specified (is not NONE), you can specify the DPC2 parameter to define a different DPC for backup routing. Similarly, when the SSN parameter is specified, you can use the SSN2 parameter to define a different SSN for backup routing.

To enter the parameters using the Terminal Handler, set the environment variable GTT_WITH_BACKUP_DPC_SSN to **1** (one) before starting the SINAP node. You can make this environment variable permanent by uncommenting it in the file $SINAP_HOME/Bin/sinap_env.[sh or csh] before starting the SINAP node:

```
GTT_WITH_BACKUP_DPC_SSN=1
```

To input the command through send_cm, use the existing syntax for CREATE-GTT, but add the fields SSN2= and DPC2= to the entry where these fields represent the alternate SCCP. It is not necessary to set this environment variable when using the send_cm command. For example,

```
send_cm -s"CREATE-GTT:GTI=4,TT=8,NP=7,NOAI=2,LADDR=500000,
HADDR=999999,SSN=254,SSN2=253,DPC=2730,DPC2=2731;"
```

NOTES

1. This command must be entered on one line only.

2. Since the parsing is position-independent, you can use a different order of keywords, for example:

```
send_cm -s"CREATE-GTT:GTI=4,TT=8,NP=7,NOAI=2,
LADDR=500000, HADDR=999999,DPC=2730,SSN=254,
DPC2=2731,SSN2=253;"
```

The sample command indicates to the SINAP/SS7 system that, for this global title, if the SCCP for DPC 2730 is out of service, the global title must be translated at the designated remote SCCP, DPC 2731. Also, if the global title has been locally translated and DPC 2730, SSN 254 is out of service, the results of the global title will be transferred to DPC 2731, SSN 253.

NOTE

Remote systems must keep the SINAP/SS7 system informed of their subsystem status by means of the appropriate messages, such as SSA/SSP, for this feature to work.

For backward compatibility, it is not necessary to specify the `DPC2` or `SSN2` fields if this feature is not desired.

If the primary and secondary SCCPs are both unavailable, the SINAP/SS7 system returns a `NOTICE` error message.

Defining and Maintaining GTT Table Entries

To configure your SINAP/SS7 node to implement GTT functionality, issue the MML command `CREATE-GTT` to define a global title entry for each global title that you plan to use or that the SINAP/SS7 system might encounter. The global title entry must describe the global title and indicate how it should be translated.

After you define global title entries for your SINAP/SS7 node, the node can support applications that implement GTT functionality. For information about the issues to consider when developing an application that implements GTT functionality, see, Defining Application Logic for Implementing GTT at the next section.

The *SINAP/SS7 User's Guide* (R8051) describes the following commands you use to define and maintain GTT table entries:

- `CREATE-GTT` creates a global title entry.
- `CHANGE-GTT` changes the contents of an existing global title entry.
- `DELETE-GTT` deletes a global title entry.
- `DISPLAY-GTT` displays all global title entries configured on your SINAP node.

Defining Application Logic for Implementing GTT

Consider the following issues as you design and develop application-programming logic for implementing GTT in an application.

- When defining the SCCP called party address, set bit 7 of the address indicator to 0 to specify routing on global title. Set bit 7 to 1 to specify routing on DPC/SSN.

  In addition, **you must set all other bits of the address indicator accordingly**. For example, if the SCCP called party address contains a DPC, you must set the PC indicator bit to 1. Likewise, if the SCCP called party address contains an SSN, you must set the SSN indicator bit to 1.

- You can include a global title in either of the following types of messages.
  - For connectionless services, include the global title in UNITDATA messages.
  - For connection-oriented services, include the global title in the connection-request message used to establish a connection with a remote application. (For information about how to code your application to implement connection-oriented services, see Implementing Connection-Oriented Services in an Application later in this chapter.)
- You can code an application to call the CASL function `ca_lookup_gt()` to perform a global title lookup and return the translation results for a specific global title.

Fictitious Originating Point Code (ANSI only)

The fictitious originating point code (FOPC) feature enables the ANSI network variant of the SINAP/SS7 system to set the MTP routing label's OPC field to an OPC that is different than the SCCP calling party address's point code. The FOPC defines the OPC that the SINAP/SS7 system is to use in place of the MTP routing label's OPC. This functionality is typically used in handover processing. Using the FOPC, the SINAP/SS7 system can set the MTP routing label's OPC field to any OPC, including the SINAP/SS7 system's own signaling point (OSP).

Three MML commands facilitate the use of an FOPC. (See the *SINAP/SS7 User's Guide* (R8051) for detailed descriptions of the commands.)

- `CREATE-FOPC` defines the FOPC you want the SINAP/SS7 system to use.
- `DISPLAY-FOPC` displays an existing FOPC.
- `DELETE-FOPC` deletes an existing FOPC.

The field, `fictitious_OPC`, is part of the `t_block_t` structure. It is a Boolean data type that indicates whether the SINAP/SS7 system is to use the FOPC feature.

- Set `fictitious_OPC` to `FALSE` if you do not want the SINAP/SS7 system to use the FOPC feature. In this case, the MTP routing label's OPC is copied from the SCCP calling party address's point code, if provided, or else from the SINAP node's own signaling point code (OSP) configured using the `CREATE-OSP` command.

- Set `fictitious_OPC` to `TRUE` if you want the SINAP/SS7 system to use the FOPC feature. In this case, the MTP routing label's OPC is derived from the FOPC created with the `CREATE-FOPC` command. Note that the calling-address-pointer field of the SCCP-level message retains a pointer to the calling party's OPC.

To use FOPC functionality in your SINAP/SS7 applications, perform the following steps.

1. Define the following environment variable and assign it the value shown in the example below. (Use whatever procedures are appropriate for the UNIX shell you are using.) You might also want to edit your SINAP/SS7 environment file (`sinap_env.sh` or `sinap_env.csh`) so that this variable is defined automatically each time you start the SINAP/SS7 system. Note that you must define the variable **before** you start the SINAP/SS7 system.

   ```
   ANSI_SINAP_FOPC=YES
   ```

2. Issue the following MML command to create the FOPC to be used in place of the MTP's routing label's OPC (where *`network-cluster-member`* defines the OPC).

   `CREATE-FOPC:FOPC=`*`network-cluster-member`*`;`

3. For a TCAP user application, before calling the `ca_put_tc()` function to send the MSU, set the tc_user.`fictitious_OPC` field of the `t_block_t` structure to `TRUE`. For a SCCP and MTP application, it must set `m_block_t structure's tc_ctrl.call_disposition` to `TRUE` to have the FOPC used in the OPC field of the MTP routing label.

Alternative Destination Point Code (ANSI, CCITT, and China only)

The alternative destination point code (ADPC) feature enables the SINAP/SS7 system to set the MTP routing label's DPC field without the provision of DPC value from the SCCP called party address. Normally, an outgoing TCAP or SCCP message specifies both the SSN and DPC (or a GT to be translated by SINAP/SS7 system into SSN and DPC) before invoking CASL API - ca_put_tc() or ca_put_msu(). The DPC of the SCCP called party address is in turn copied to the DPC field of the MTP routing label for outbound routing purpose. With ADPC feature, the application can send an outgoing TCAP or SCCP message with no DPC provided at the SCCP called party address, which is useful for processing a handover query. In this case, SINAP TCAP or SCCP application needs to specify the alternative DPC so that SINAP/SS7 system can set the DPC field of the MTP routing label to the value defined by the alternative DPC feature.

1. To use the alternative DPC feature in the ANSI or China TCAP variant, mask the `t_block_t` structure's `tc_user.thp.tb_options` with `USE_ALT_DPC (0x01`, that is, mask bit position 1 to 1) and set:
   - `tc_user.thp.alt_DPC[0]` to the member field of the alternative DPC
   - `tc_user.thp.alt_DPC[1]` to the cluster field of the alternative DPC
   - `tc_user.thp.alt_DPC[2]` to the network field of the alternative DPC

2. To use the alternative DPC feature in the CCITT TCAP variant, mask the `t_block_t` structure's `tc_user.dhp.tb_options` with `USE_ALT_DPC (0x01`, that is, mask bit position 1 to 1) and set `tc_user.dhp.alt_DPC`, which is `U32` (32-bit unsigned integer). The CCITT TCAP variant references the `tc_user.dhp` structure (dialogue handling primitive).
3. The ANSI or China variant's SCCP user application references the tc_alt.alt_dpc structure. The data structure contains four U8 fields for member, cluster and network of the ANSI or China point code format and a status field. To use the alternative DPC feature in the ANSI or China variant, the SCCP user application set the m_block_t data structure's tc_alt.alt_dpc.status with USE_ALT_DPC (0x01) and set:
   - tc_alt.alt_dpc.member to the member field of the alternative DPC
   - tc_alt.alt_dpc.cluster to the cluster field of the alternative DPC
   - tc_alt.alt_dpc.network to the network field of the alternative DPC
4. The CCITT variant's SCCP user application references the tc_alt.alt_ccitt structure. The data structure contains a U16 dpc field and a U8 status field. To use the alternative DPC feature in the CCITT variant, the SCCP user application set the m_block_t data structure's tc_alt.alt_ccitt.status with USE_ALT_DPC (0x01) and set tc_alt.alt_ccitt.dpc to the alternative DPC value.

Enhanced Message Distribution

Enhanced message distribution is available to applications that interface with the SINAP/SS7 system at the SCCP or the TCAP boundary. This feature provides a mechanism for expanding the discrimination rules that the SINAP/SS7 system uses to route incoming MSUs to their destinations. For example, an application can use this feature to:

- Accept input for a number of other applications
- Accept input from a specific OPC or set of OPCs
- Use the same SSN as another application

The SINAP/SS7 system routes an incoming MSU to its destination based on the MSU's SSN. If an application is registered with that SSN, the SINAP/SS7 system routes the incoming MSU to that application. An application that implements enhanced message distribution can define additional criteria that must be met in order for the SINAP/SS7 system to route incoming MSUs to it. These criteria are referred to as an application's *message distribution information*.

When an application implements enhanced message distribution, the SINAP/SS7 system examines every incoming MSU destined for the application. If the characteristics of the incoming MSU match the application's message distribution information, the SINAP/SS7 system passes the MSU to the application; otherwise, if it is configured to do so, the SINAP/SS7 system discards the MSU.

Processing Overview

The CASL function `ca_dist_cmd()` and the structure `dist_cmd_t` are used to facilitate enhanced message distribution.

- The `ca_dist_cmd()` function provides access to message distribution information. An application calls this function to define message distribution information, and to retrieve or delete existing message distribution information.
- The structure `dist_cmd_t` is used to pass message distribution information. When the `ca_dist_cmd()` function is called, a pointer to this structure is passed to the function. The structure's `cmd` field specifies whether the `ca_dist_cmd()` function call is to define, retrieve, or delete message distribution information for an application. If the function call is to define message distribution information, this structure contains that information; if the function call is to retrieve message distribution information, this is the structure to which that information is written.

  The `dist_cmd_t` structure contains two arrays: one lists the SSNs considered valid for the application, the other lists valid OPCs. The SINAP/SS7 system uses these arrays to determine whether to route an incoming MSU to the application.

To implement enhanced message distribution, an application registers with the SINAP/SS7 system with certain `register_req_t` structure fields set to specific values. These values tell the SINAP/SS7 system that the application is implementing enhanced message distribution. The application defines its message distribution information by initializing the `dist_cmd_t` structure and calling the CASL function `ca_dist_cmd()`, passing a pointer to this structure. Thereafter, the SINAP/SS7 system routes incoming MSUs to the application based on the application's message distribution information.

The following is a list of enhanced message distribution features.

- An application's message distribution information can be updated dynamically during SINAP/SS7 operation without disrupting the application's active processing.
- If an application has concerned point codes (CPCs), the SINAP/SS7 system maintains status information for each CPC.
  - If an application's status changes, the SINAP/SS7 system notifies each CPC associated with that application.
  - If a CPC's status changes, the SINAP/SS7 system notifies each application associated with that CPC.
- The SINAP/SS7 system provides two environment variables for specifying how the SINAP/SS7 system is to handle discarded MSUs.
  - `DISCARDS_PER_ALARM` specifies the number of MSUs that the SINAP/SS7 system is to discard before generating an alarm.
  - `UDTS_NO_OPC` specifies whether the SINAP/SS7 system is to generate a UDTS (UnitData Service) message when the MSU's OPC is not valid for the specified SSN.

The Message Distribution Information Structure

The structure in which message distribution information is stored, `dist_cmd_t`, has the following format. The `dist_cmd_t` structure is defined in the include file `register.h`, as are the variables `MAX_APPL_SSN` and `MAX_APPL_OPC`.

```
typedef struct dist_cmd_s
{
      U32 appl;                /* APPL_THIS -1 */
      U8  cmd;                 /* DIST_SET 1 */
                               /* DIST_DEL 2 */
                               /* DIST_INQ 3 */
      U8  boundary;            /* SS7_INPUT_BOUNDARY_NA 0 */
                          /* SS7_INPUT_BOUNDARY_SCCP23 4 */
      S8  ssn_count;           /* DIST_ALL 0 */
      U8  opc_count;           /* DIST_ALL_OTHER 0 */
      U8  ssn[MAX_APPL_SSN];   /* 32 */
      U32 opc[MAX_APPL_OPC];   /* 128*/
} dist_cmd_t;
```

Table 3-16 provides a brief description of the fields in the `dist_cmd_t` structure. Detailed information about these fields is presented in the section Defining Message Distribution Information later in this chapter.

Table 3-16. `dist_cmd_t` Structure Fields

Field	Description
`appl`	Specifies the name of the application whose message distribution information is being defined or retrieved
`cmd`	Defines the task to perform on the application's message distribution information: define, delete, or retrieve
`boundary`	Defines the boundary on which task is registered: `SS7_INPUT_BOUNDARY_NA` for non-COF or `SS7_INPUT_BOUNDARY_SCCP23` for COF. Only required for `DIST_INQ` with `appl = 0.`
`ssn_count`	Specifies the number of SSNs to associate with the application
`opc_count`	Defines the number of OPCs from which the application can accept input
`ssn`	An array of SSNs, each of which will be associated with the application
`opc`	An array of OPCs, from each of which the application can accept input

Implementing Enhanced Message Distribution

This section provides information about how to implement enhanced message distribution for an application. It contains the following subsections.

- "Considerations" describes things you should be aware of as you implement enhanced message distribution.
- "Handling Discarded MSUs" provides instructions for specifying how you want the SINAP/SS7 system to handle discarded MSUs.
- "Application Registration" describes the procedure the application must follow to register with the SINAP/SS7 system.
- "Defining Message Distribution Information" describes the procedure the application must follow to define message distribution information.
- "Activating the Application" provides instructions for activating the application.

Considerations

You should consider the following when implementing enhanced message distribution for a SINAP/SS7 application.

- Enhanced message distribution increases the demand on shared-memory usage; therefore, you may need to increase the value of the UNIX tunable system parameter `SHMMAX`.

  For the HP-UX operating system, use the HP-UX operating system administration utility (`sam`) to change the `SHMMAX` and `SHMSEG` system parameters in the `/usr/conf/master.d/core-hpux` file.

  Before modifying the `SHMMAX` parameter, you should check with your system administrator to make sure the new value will work with other products and applications running on the system. See the appropriate documentation for your UNIX system for more information.

- If the application uses the load control facility, be aware of the following:
  - When the *SINAP/SS7 User's Guide* (R8051) instructs you to specify the application's SSN for load control MML commands and CASL functions, you must specify the application's name instead (for example, `ENABLE-LOAD-CONTROL,SSN=`**`DB12`** instead of `ENABLE-LOAD-CONTROL,SSN=`**`230`**).
  - In addition, for CASL you must specify the application name as a zero filled, right justified U32 word. (For instructions on how to convert the application name to this format, see the description of the `appl` field in the section, "Defining Message Distribution Information," later in this chapter.)

Handling Discarded MSUs

The SINAP/SS7 system provides two environment variables for specifying how you want the SINAP/SS7 system to handle discarded MSUs. Define these environment variables at a UNIX

system prompt **before** you start the SINAP/SS7 system by following the appropriate procedure for the shell you are using.

- Define the following environment variable if you want the SINAP/SS7 system to generate a UDTS message when it receives an MSU whose OPC is not valid for the specified SSN. If you do not define this variable, the SINAP/SS7 system discards the MSU and does not generate a UDTS message. You need not assign a value to the variable; the SINAP/SS7 system simply checks that the variable exists.

  ```
  UDTS_NO_OPC
  ```

- Define the following environment variable if you want the SINAP/SS7 system to generate an alarm after discarding the number of MSUs defined by *n*. If you do not define this environment variable, the SINAP/SS7 system generates an alarm after discarding approximately five MSUs (this number may vary).

  ```
  DISCARDS_PER_ALARM=n
  ```

To turn off the functionality implemented by either environment variable, remove the variable's definition by following the appropriate procedure for the shell you are using. You must then restart the SINAP/SS7 system for the changes to take effect. (For instructions, see the UNIX documentation for that shell.) For example, if you are using the C shell, issue the command `unsetenv UDTS_NO_OPC` and restart the SINAP/SS7 system.

Application Registration

To register with the SINAP/SS7 system, an application initializes the fields of the `register_req_t` structure and then calls the CASL function `ca_register()`. To implement enhanced message distribution, an application must follow certain guidelines when registering with the SINAP/SS7 system. For example, the application must initialize certain fields of the `register_req_t` structure to specific values. (For more information about application registration and the `register_req_t` structure, see the description of the `ca_register()` function in
Chapter 6.)

> NOTE
>
> The application must initialize all `register_req_t` fields before calling `ca_register()` to register with the SINAP/SS7 system.

To implement enhanced message distribution, the application must follow these guidelines when registering with the SINAP/SS7 system.

- If the application is configured for load control, the application's name, which is specified in the `appl` field, must contain at least two alphabetic characters: `a` through `z` or `A` through `Z` (for example, `DB12` or `TCRV`). These alphabetic characters are needed so that the application implements enhanced message distribution **and** uses the load control facility.

See the section Considerations earlier in this chapter for additional guidelines and considerations.

NOTE

In the `register_req_t` structure's `appl` field, the application name is specified as an ASCII character string of up to four bytes. However, the application name must be specified as a zero filled, right justified, U32 word in the `appl` field of the `dist_cmd_t` structure and in CASL functions used to implement load control. For information about how to convert the application name to and from this format, see the descriptions of the `ca_pack()` and `ca_unpack()` functions in Chapter 6, ‘‘CASL Function Calls.”

- The application must register to receive input at the SCCP or TCAP boundary. To do this, the application must specify one of the values listed in Table 3-17 for the `ss7_input_boundary` fields. Applications that receive input at the MTP boundary cannot implement enhanced message distribution.

Table 3-17. SS7 Input Boundary Settings for Enhanced Message Distribution

SS7 Input Setting	Description
`SS7_INPUT_BOUNDARY_SCCP=2`	UDT SCCP
`SS7_INPUT_BOUNDARY_TCAP=3`	UDT TCAP
`SS7_INPUT_BOUNDARY_SCCP23=4`	SCCP COF
`SS7_INPUT_BOUNDARY_TCAPX=6`	XUDT TCAP
`SS7_INPUT_BOUNDARY_SCCPX=7`	XUDT SCCP

NOTE

It is possible to have two applications with the same point code and SSN, one registered at the SCCP boundary and the other at the SCCP23 boundary, handling traffic for both connectionless (SCCP) and connection oriented (SCCP23) service. Both of these processes are coded to use Enhanced Distribution. Both applications share the same SSN and may have identical OPC lists. Support is limited to one SSN only.

- The application must initialize the `sio_ssn_ind` and `ssn_sio` fields of the `register_req_t` structure to the following values. These values indicate that the application will have multiple SSNs and/or multiple OPCs associated with it.

  - For `sio_ssn_ind`, specify the value `REG_MULT`.
  - For `sio_ssn`, specify the value `0`.

- All application instances must register with the same set of registration parameters (for example, `sio_ssn` or `sio_ssn_ind`).
- All application instances must use the same message distribution information. The SINAP/SS7 system does not allow an application instance to register if its message distribution information differs from that of an already registered application instance.
- The application can have a maximum of two process names registered with the SINAP/SS7 system.

Defining Message Distribution Information

To define message distribution information, an application must initialize the fields of the `dist_cmd_t` structure and then call the `ca_dist_cmd()` function, passing a pointer to this structure. The following list describes each field in the `dist_cmd_t` structure.

- The `appl` field specifies the name of the application whose message distribution information is being defined or retrieved. In the `register_req_t` structure's `appl` field, the application name is specified as an ASCII character string of up to four bytes. However, in the `appl` field of the `dist_cmd_t` structure, the application name must be specified as a zero filled, right justified, U32 word. To convert an application name to this format, code the application so that it calls the function `ca_pack()`, passing the character string that defines the application name; the function returns the application name as a zero-filled, right-justified, U32 word. To convert the application name back to a character string, code the application so that it calls the function `ca_unpack()`, passing the U32 word and a pointer to a character string; the function converts the application name to a character string.
- The `cmd` field specifies the task to perform on the application's message distribution information. Valid values are as follows:
  - `DIST_SET` defines the message distribution information for an application or modifies an application's existing message distribution information.
  - `DIST_DELETE` deletes an application's message distribution information, which means that the application no longer supports enhanced message distribution.
  - `DIST_INQ` retrieves an application's message distribution information.
- The boundary field is only required for the `DIST_INQ` command when the appl field is 0. It should be set to `SS7_INPUT_BOUNDARY_NA` to select a non-COF application and to `SS7_INPUT_BOUNDARY_SCCP23` to select a COF application.

The remaining fields: `ssn_count`, `ssn`, `opc_count`, and `opc` define the discrimination rules that you want the SINAP/SS7 system to follow when routing incoming MSUs to the application. See the subsections "Applications Using SSN Discrimination," "Applications Using OPC Discrimination," and "Applications Using the Same SSN" later in this section for additional information about how to define different types of discrimination rules for an application.

Use the `ssn_count` and `ssn` fields to define an SSN array that lists the SSN(s) that you want to associate with the application. When the SINAP/SS7 system receives an incoming MSU, it examines the MSU's SSN; if the SSN is listed in the application's SSN array, the SINAP/SS7 system sends the MSU to the application. If you do not want the SINAP/SS7 system to perform message discrimination based on an incoming MSU's SSN, specify the value `0` for `ssn_count` and leave the SSN array empty.

- The `ssn_count` field specifies the number of SSNs that you want to associate with the application. This value cannot exceed the value of `MAX_APPL_SSN`, which is defined in the include file `$SINAP_HOME/Include/register.h`. The SSN array defined by the `ssn` field must contain as many entries as `ssn_count` defines.
- The `ssn` field is an array of SSNs, each of which is to be associated with the application. The number of SSNs in this array must match the number defined by `ssn_count`. If the value of `ssn_count` is `0`, make sure that this array is empty.

Use the `opc_count` and `opc` fields to define an OPC array that lists the OPC(s) from which you want the application to receive incoming MSUs. When the SINAP/SS7 system receives an incoming MSU destined for the application, it examines the MSU's OPC; if the OPC is listed in the application's OPC array, the SINAP/SS7 system sends the MSU to the application. If you do not want the SINAP/SS7 system to perform message discrimination based on an incoming MSU's OPC, specify the value `0` for `opc_count` and leave the OPC array empty.

- The `opc_count` field specifies the number of OPCs from which you want the application to accept input. This value cannot exceed the value of `MAX_APPL_OPC`, which is defined in the include file `$SINAP_HOME/Include/register.h`. The OPC array defined by the `opc` field must contain as many entries as `opc_count` defines.
- The `opc` field is an array that lists each OPC from which the application can accept input. The number of OPCs in this array must match the number defined by `opc_count`. If the value of `opc_count` is `0`, make sure that this array is empty.

Applications Using SSN Discrimination

If an application's message distribution information includes an SSN array, the SINAP/SS7 system uses *SSN discrimination* to route incoming MSUs to the application: if the MSU's SSN is listed in the SSN array, the SINAP/SS7 system routes the MSU to the application. To configure the application to accept incoming MSUs for a number of other applications, define an SSN array that lists the SSN of each of these applications. When the SINAP/SS7 system receives an incoming MSU destined for one of these applications, the SINAP/SS7 system routes the MSU to this application, which is then responsible for delivering the MSU to the appropriate SSN.

NOTE

You can also define a list of OPCs from which the application can receive input by creating an OPC array. In this case, the SINAP/SS7 system routes an incoming MSU to the application only if both the MSU's SSN and OPC are listed in these arrays.

If registered at the SCCP23 boundary only one SSN is supported in the SSN array. There may be multiple OPCs in the OPC array.

The following sample `dist_cmd_t` structure contains the message distribution information that you might define for the application `HNDL`, which accepts input for three SINAP/SS7 applications (SSNs `120`, `220`, and `240`).

```
typedef struct dist_cmd_s
{
    U32    HNDL;          /* appl */
    U8     DIST_SET;      /* cmd */
    U8     0;             /* boundary */
    S8     3;             /* ssn_count */
    U8     0;             /* opc_count */
    U8     [120           /* ssn */
            220           /* ssn */
            240];         /* ssn */
    U32    [NULL];        /* opc */
} dist_cmd_t;
```

Applications Using OPC Discrimination

If an application's message distribution information includes an OPC array, the SINAP/SS7 system uses *OPC discrimination* to route incoming MSUs to the application: if the MSU's OPC is listed in the OPC array, the SINAP/SS7 system routes the MSU to the application. If the OPC is not listed in the array, the SINAP/SS7 system discards the MSU; however, if you have defined the environment variable `UDTS_NO_OPC`, the SINAP/SS7 system does not discard the MSU, but instead sends a UDTS message to the OPC.

Applications Using the Same SSN

Sometimes it is useful for multiple applications to use the same SSN. For example, to have the SINAP/SS7 system interface with a heterogeneous group of network switches, you can develop several similar SINAP/SS7 applications, each of which supports a particular type of network switch. By associating a specific OPC or set of OPCs with each application process, you can direct input from a network switch to the appropriate switch-supporting application process.

NOTE

When multiple applications use the same SSN, each application **must** define an OPC array that specifies a different OPC or set of OPCs; applications that use the same SSN **cannot** accept input from the same OPC.

If several applications use the same SSN, each application must initialize the following fields of its `dist_cmd_t` structure as follows:

- For `appl`, specify the name of the application (a zero-filled, right-justified, U32 word), which must be unique.
- For `ssn`, define an array that lists the SSN being used, which will be the same for each application (`ssn_count` will be `1`).
- For `opc`, define an array that lists a unique OPC or set of OPCs from which the application can accept input (`opc_count` will specify the number of OPCs in this array).

The following sample `dist_cmd_t` structure shows the message distribution information for three SINAP/SS7 applications, each of which uses the SSN `230`. The first application, `SW1`, accepts input from OPC `254-052-001`; the second, `SW2`, accepts input from OPC `254-020-005`; and the third, `SW3`, accepts input from OPC `230-002-001`.

```
{                                   {
 appl      = SW1;                    appl      = SW2;
 cmd       = DIST_SET;               cmd       = DIST_SET;
 boundary  = 0;                      boundary  = 0;
 ssn_count = 1;                      ssn_count = 1;
 opc_count = 1;                      opc_count = 1;
 ssn       = 230;                    ssn       = 230;
 opc       = 254-052-001;            opc       = 254-020-005;
}                                   }

{
 appl      = SW3;
 cmd       = DIST_SET;
 boundary  = 0;
 ssn_count = 1;
 opc_count = 1;
 ssn       = 230;
 opc       = 230-002-001;
}
```

Activating the Application

To activate an application that implements enhanced message distribution, code the application so that it sends an `I_N_STATE_REQ`, `SCMG_UIS` message to the SCCP management process (SCMG) for each of its configured SSNs. This step activates the application, which now implements enhanced message distribution.

NOTE

Before going out of service, have the application send an `I_N_STATE_REQ`, `SCMG_UOS` message to the SCMG for each of its configured SSNs.

Retrieving Message Distribution Information

You can retrieve two types of message distribution information: the SSN and OPC arrays associated with a particular application, or the name of the application whose message distribution information matches a particular SSN/OPC combination. To retrieve message distribution information, code the application so that it calls the `ca_dist_cmd()` function and passes a pointer to a `dist_cmd_t` structure; the SINAP/SS7 system returns the requested information in this structure.

- To retrieve the SSNs and OPCs associated with a particular application, initialize the `dist_cmd_t` structure's `appl` field to the name of the application (a zero-filled, right-justified, U32 word). The SINAP/SS7 system returns the application's SSN and OPC arrays in the `dist_cmd_t` structure's `ssn` and `opc` fields.
- To retrieve the name of the application whose SSN and OPC criteria match a particular SSN/OPC combination, initialize the `dist_cmd_t` structure's `appl` field to the value `0` and initialize the `ssn[0]` and `opc[0]` fields to the SSN/OPC combination. You must also specify the application's boundary, either `SS7_INPUT_BOUNDARY_NA` to select a non-COF application or `SS7_INPUT_BOUNDARY_SCCP23` to select a COF application. The SINAP/SS7 system returns the name of the application (a zero-filled, right-justified, U32 word) in the `appl` field of the `dist_cmd_t` structure. This is the application to which the SINAP/SS7 system would send an incoming MSU with that particular SSN/OPC combination.

Changing Message Distribution Information

You can change an application's message distribution information without disrupting the application's active processing. When you change an application's message distribution information, the changes apply to each of the application's instances. Note, however, that these changes do not affect an application's existing transactions (those that are waiting in an input queue). This is because existing transactions are considered to have already been delivered.

To change message distribution information, an application must first retrieve the existing information, then modify it, and save the changes. To do this, code the application so that it performs the following steps.

1. Initialize the fields of the `dist_cmd_t` structure as follows:
   - Initialize `appl` to the name of the application (a zero-filled, right-justified, U32 word) whose message distribution information you want to change.
   - Initialize `cmd` to the value `DIST_INQ`.
2. Call the `ca_dist_cmd()` function and pass it a pointer to the `dist_cmd_t` structure that was initialized in Step 1. The SINAP/SS7 system returns the application's message distribution information in the `ssn_count`, `opc_count`, `ssn`, and `opc` fields of the structure.
3. Initialize the `dist_cmd_t` structure to modify the application's message distribution information and initialize the `cmd` field to the value `DIST_SET`.

4. Call `ca_dist_cmd()`, passing a pointer to the structure that was initialized in step 3. The function call updates the application's message distribution information to reflect the changes to `dist_cmd_t`.

Deleting Message Distribution Information

When you delete an application's message distribution information, the application no longer supports enhanced message distribution. To delete an application's message distribution information, code the application so that it calls the `ca_dist_cmd()` function, passing a pointer to a `dist_cmd_t` structure whose fields are initialized as follows:

- Initialize `appl` to the name of the application (a zero-filled, right-justified, U32 word) whose message distribution information you want to delete.
- Initialize `cmd` to the value `DIST_DEL`.

You cannot delete a portion of an application's message distribution information by following this procedure. Instead, you must follow the procedure described in the preceding section, "Changing Message Distribution Information."

SCCP Third Party Address

This field facilitates routing between an SSP/node, an agent, and an application. The field saves the original routing information and replaces it with the routing information of the TCP/IP agent. The SINAP/SS7 system stores information specified in the `sccp_3rd_party_addr` field in the `mblock` as the original calling party address information so the information is retained when an SS7 over TCP/IP agent (registered at the SCCP boundary) overwrites the original SCCP called party address with its own point code and pseudo SSN. The information is required to establish a two-way dialogue with an application registered at the TCAP boundary on the same SINAP node and system. When the TCP/IP agent receives messages from a TCAP application, the agent requires the original SCCP calling party address to correctly format and route messages back to the originating node over TCP/IP.

For a TCAP application (accessed through the TCP/IP agent) originating a dialogue (for CCITT variants) or transaction (for ANSI variants), the field specifies the SCCP called party address of the TCAP application. In this case, the called party address is required because the original called party address provided in the `tblock` and `mblock` is configured to address the own signaling point (OSP) code and pseudo SSN of the TCP/IP agent running on the same SINAP node.

The CASL transparently copies the `sccp_3rd_party_addr` field between the `tblock` and `mblock` in both directions when sending and receiving `tblocks`. The SINAP driver initializes this field in the `mblock` to zeros when the SINAP node receives messages from Level 2 of the SS7 network.

The constant specified in the `MAX_ADDR_LEN` parameter is defined in the `tblock.h` and `mblock.h` include files.

Custom Application Distribution

The SINAP/SS7 system supports a custom application distribution (CAD) feature that enables a SINAP node to distribute TCAP message traffic to specific applications based on the value of selected, protocol-specific, message parameters. The CAD feature is implemented on one or more SINAP nodes as an extension to the capabilities provided by enhanced message distribution (multiple applications registered for the same SSN).

CAD currently provides a protocol-specific implementation based on the European Telecommunications Standards Institute (ETSI) Capability Set 1 (CS-1) Intelligent Network Application Protocol (INAP) standards. (ETSI CS-1 INAP is derived from the ITU-T CS-1 INAP standards which can also be supported by this feature.)

CAD requires much of the same functionality provided by the SINAP/SS7 enhanced message distribution feature. This includes the capability to filter and distribute message traffic based on OPCs as well as SSNs. (See the section on Enhanced Message Distribution earlier in this chapter.) Like enhanced message distribution, CAD is available to applications that interface with the SINAP/SS7 system at the TCAP boundary. CAD provides an additional mechanism (ServiceKeys) for further expanding the discrimination rules that the SINAP/SS7 system uses to route incoming MSUs to their destinations.

Generic CAD Registration

Applications requiring custom distribution must initially register with the SINAP node to receive input at the TCAP boundary and specify the value `REG_MULT` for the `sio_ssn_ind` and `0` for the `sio_ssn` fields in the `register_req_t` structure. Except for being restricted to the TCAP boundary, this is the same procedure currently used for applications that register for enhanced message distribution.

Use the function `ca_cust_dist_cmd()`to specify the custom application distribution criteria. This function is an extended version of the `ca_dist_cmd()` function used for enhanced distribution. In addition to the `dist_cmd_t` structure used to specify the application name, command, SSN, and OPC criteria, the `ca_cust_dist_cmd()` function also specifies the ID of the type of custom distribution being implemented and a type-specific structure used to define the custom distribution criteria. The type-specific criteria structure is specified as an abstract (`void *`) pointer. The actual structure type is determined by the custom type ID parameter.

Unlike enhanced message distribution, multiple applications can be registered for the same SSN and OPC, as long as they all use the same custom distribution type ID. As with enhanced distribution, you can implement custom distribution for all SSNs, or for all OPCs with one or more SSNs. Although you can change the custom distribution criteria for an application at run-time, you cannot change the custom distribution type, once it has been set.

CS-1 INAP-Specific CAD Registration

Applications requiring Capability Set 1 (CS-1) Intelligent Network Application Protocol (INAP) custom distribution are able to specify ServiceKey values in addition to SSN and OPC values when specifying their custom application distribution criteria within the

`ca_cust_dist_cmd()` function. The custom type-specific criteria structure includes an integer ServiceKey Count field and a fixed-size integer ServiceKey array allowing up to 64 entries (per SINAP node). An application may specify from 0 to 64 ServiceKey values.

An application that registers for CS-1 INAP custom application distribution with `a` ServiceKey value of `0` (zero) is designated as the default or fallback application. All message traffic that matches the SSN and OPC criteria specified for that application, but cannot be sent to any other application registered for the same SSN or OPC with specific ServiceKey criteria, is sent to the fallback application. This includes TCAP `TC-BEGIN` messages that contain:

- InitialDP invoke operations without a ServiceKey parameter
- InitialDP invoke operations with a ServiceKey value not specified by any other application
- Any invoke operation other than InitialDP or AssistRequestInstructions (ARI)
- Any message that does not meet the parsing criteria applied when determining the component type, operation, ServiceKey, or Correlation ID value, but does match the specified SSN/OPC criteria

The `ca_cust_dist_cmd()` parameter criteria with ServiceKey values of `0`, also enables CS-1 INAP ARI processing for applications that do not require the ServiceKey based application distribution feature.

NOTES

1. Only one service may register as the fallback application for a given SSN/OPC criteria.
2. Two applications may not register for the same ServiceKey and the same SSN/OPC criteria.
3. No specific custom distribution parameters are required in relation to CS-1 INAP ARI operation processing.

Generic CAD Message Processing

Once the SINAP/SS7 system determines that `REG_MULT` is specified for the SSN of a received message, enhanced message distribution processing is invoked. If enhanced distribution determines that the SSN and/or OPC entry appropriate for the received message contains a custom application distribution type ID, instead of an application index, the SINAP/SS7 system checks a type-specific distribution table to determine the destination of the message. The destination of the message may be an application, a process, or undefined, if the message is to be discarded. The SINAP/SS7 system may also modify fields in the `mblock` structure containing the message. Specifically, the SINAP/SS7 system fills in the `pid` of the process to receive the message, if the message is to be distributed directly to a specific process. When the SINAP node determines the destination of the message is an application, the node uses the load control type registered for that application to determine the final destination process.

The SINAP node only invokes the custom type-specific distribution function for TCAP `BEGIN` (`QUERY` message in ANSI) or `UNI` messages. Other message types are always distributed to the process that is already handling the established dialogue (or transaction in ANSI). The SINAP/SS7 system determines if a given message is a TCAP message, as opposed to an SCCP user part message of some other type, before it can determine the TCAP package type and custom type-specific distribution. The fact that custom application distribution is specified for that SSN/OPC implies that TCAP parsing is appropriate.

CS-1 INAP Message Processing

The SINAP node parses all messages sent to the CS-1 INAP type-specific distribution function to confirm that the first message component is an invoke component and to extract the operation code. Both the CCITT and ANSI forms of the operation code are supported. Any failure to decode these fields results in fallback message handling. This applies whenever errors occur during the parsing of the message. Further processing of the message is determined by the operation code. The SINAP/SS7 system implements specific processing for only the `InitialDP` and `ARI` operations. Any other operation results in fallback message handling.

CS-1 INAP Message Processing For an InitialDP Operation

Before any further message parsing is performed, the SINAP/SS7 system takes the following actions:

- Determine if any ServiceKeys are configured for the specified SSN/OPC.

  If yes, parsing continues.

  If not, the SINAP node performs fallback message handling.

- The SINAP node confirms the presence of a parameter sequence tag followed by a parameter sequence length.
- The following tag must be that of the ServiceKey parameter `[CONTEXT 0]`, or the SINAP node assumes that no ServiceKey parameter is present, and it performs fallback message handling.
- The ServiceKey parameter length and value is then decoded.
- The SINAP/SS7 system's table of ServiceKey values configured for the specified SSN/OPC is then searched.

  If a match is found, the table index for the application specified in the ServiceKey table entry is returned, along with the indication that the message should be distributed to an application.

  If no matching ServiceKey value is found, the SINAP node performs fallback message handling.

CS-1 INAP Message Processing For an ARI Operation

Message parsing continues with confirming the presence of the parameter sequence tag and length fields. The following tag must be that of the `CorrelationID` parameter `[CONTEXT`

`0]` or the message is considered to be in error, and fallback message handling is performed. At this point the number of octets specified by the length field are decoded as the `CorrelationID` parameter. The `CorrelationID` parameter is defined to be in the ITU-T Q.763 "Generic Digits" format. Of all the fields defined in the "Generic Digits" format, only the actual digits field is of significance to CS-1 INAP message processing. Exactly 10 binary coded decimal (BCD) digits must be present in the digits field. The SINAP node decodes the BCD digits as two integers, a five digit IPC index, and a five-digit TCAP dialogue/transaction ID. Only the IPC index is significant to the SINAP/SS7 system. The decoded IPC index is stored in the `mblock tc_ctrl ipc_index` field, and is subsequently used to look up the process ID in the IPC table. The process ID, in turn, is stored in the `mblock ca_ctrl pid` field, and the function returns an indication that the message should be routed directly to a process.

Except for requiring that the SSN/OPC enhanced distribution tables reflect the fact that CS-1 INAP type custom application distribution was specified by an application, no specific tables are maintained for ARI operation processing. The SINAP/SS7 system determines the process ID of the process that is to receive the ARI message from the IPC index embedded in the ARI `CorrelationID` parameter. This parameter, received from an intelligent peripheral (IP) with intelligent network (IN) capabilities (or an assisting SSP), by definition, contains the same digits specified in the `CorrelationID` parameter of the EstablishTemporaryConnection (ETC) operation. These are the same digits sent by the original Service Control Function (SCF) service that initiated the user-assisted "switch-out."

For example, the original switch (SSP) processing a call switches the call to another switch. (This could be another SSP or SS7 ETSI/CS-1 INAP capable intelligent peripheral (IN IP). The call is switched in order to play announcements or otherwise interact with the user. This is usually when the original SSP does not have the required announcements or capabilities to play them (under the direction of the SCF service via the ETC operation). The assisting SS7 SP (SSP or IN IP) initiates an entire new dialogue to inquire what user interaction is to be performed.

In order to more efficiently correlate the ARI to the original service that sent the ETC, the information included in the `CorrelationID` parameter must be sufficient to identify both the process where the original service is running, and the TCAP dialogue or transaction ID that identifies the specific service instance within that process.

The CASL function `ca_enc_cs1_corrid()` ensures that the digits used, when formatting the `CorrelationID` parameter, identify the IPC index and dialogue/transaction ID in the same format understood by the CS-1 INAP specific message distribution function in the SINAP/SS7 system. Applications that implement any of the features of CS-1 INAP specific custom application distribution must use this function to format the ETC `CorrelationID` parameter.

A complimentary CASL function `ca_dec_cs1_corrid()` decodes the digits field in a received ARI `CorrelationID` parameter. In this case, it is the dialogue or transaction ID that is significant to the process and must correlate the ARI to the original service instance.

If an application requires a different format of the ETC `CorrelationID` parameter digits, the application may not specify CS-1 INAP-specific custom application distribution and no other applications can be configured for the same SSN/OPC criteria.

CS-1 INAP Message Processing For SFR Operation

The SINAP/SS7 system provides specific processing of the ServiceFilteringResponse operation and treats this operation as an unrecognized operation. In this case the SINAP node performs fallback message handling.

CS-1 INAP Fallback Message Processing

Fallback message processing is invoked whenever CS-1 INAP specific message processing is unable to determine an application or process to distribute a given message to, for any reason, including errors detected in the message format. The CS-1 INAP specific routing tables include a field where for every configured SSN/OPC a fallback application, as described for CS-1 INAP specific custom application distribution registration, can be stored. If a fallback application is defined for the SSN/OPC of a message falling into this category, the message is distributed to that application. If a fallback application is not defined, an indication is returned specifying that the message should be discarded. At this point, the same processing of discarded messages employed in enhanced message distribution is utilized. This includes the environment variable tunable options to return such messages on error and/or report alarms for every `n` number of messages discarded.

The purpose of a fallback application can vary with the implementation. For example, a fallback application may be specified as the only application for a given SSN and/or OPC criteria. This would be appropriate for an application that does its own ServiceKey dispatching to internally defined services, but requires the ETSI CS-1 INAP specific ARI message processing. This is in order to ensure that the ARI messages are distributed to the same data processes that are already processing the original service. Alternately, where a separate application is defined for each of several different sets of ServiceKeys, all using the same SSN/OPC criteria, the fallback application might be defined only to process protocol errors or other unexpected message traffic. This would allow error responses, appropriate to the specific protocol, to be sent back to the originator of the offending message.

The fallback application is also the only application that can receive SFR operations, or `InitialDP` operations that do not contain a ServiceKey parameter.

SCCP Management Considerations for CAD

For the purposes of SCCP management, custom application distribution is identical to enhanced distribution. SCCP management is only concerned with the disposition of the SSN and OPC criteria.

Configuring Multiple Link Congestion Levels

The SINAP/SS7 network variants provide congestion handling in different ways, depending on the network variant. This section describes how to configure the CCITT, China, ANSI, TTC, and NTT network variants to handle network congestion.

Variant Differences

In the CCITT and China network variants, you can configure the following congestion levels:

1. International One Congestion Onset, One Congestion Abatement level (the default if the environment variable `CCITT_CONGESTION_OPTION` is not defined)
2. National Multiple Congestion States With Congestion Priority
3. National Multiple Congestion States Without Congestion Priority

In the ANSI, TTC, and NTT variants, the SINAP/SS7 system automatically implements multiple congestion levels (0-3) *with* congestion priority. The congestion priority is set within the client applications.

Congestion States

Multiple link congestion states enable the SINAP/SS7 system to maintain up to four levels of signaling link congestion (0, 1, 2, and 3), and to set a link's congestion status according to these levels. The system uses the same congestion onset, abatement, and discard levels in all variants of the SINAP/SS7 system.

The SINAP/SS7 system implements multiple link congestion levels on a system-wide basis so that when you specify the thresholds for each link-congestion level, the SINAP/SS7 system monitors each of its configured links for these thresholds. A link's congestion status indicates the level of congestion that the link is experiencing based on the number of messages on the link's SS7 driver queue. When the number of messages on the queue exceeds the number of messages allowed for a particular congestion level, the SINAP/SS7 system increases the value of the link's congestion status to indicate the level of congestion the link is experiencing. As the link becomes less congested, the SINAP/SS7 system decrements the value of the link's congestion status.

NOTE

In the variant, if the congestion priority level (set within the application) is greater than the congestion discard level set for a DPC, the message is sent. If the congestion priority level is less than the discard level, the system discards the message. The default message priority level is 0. (See the *SINAP/SS7 User's Guide* (R8051) for information on changing the message priority within an application.)

In all network variants of the SINAP/SS7 system you can display and change the settings of threshold values by issuing the MML commands, `DISPLAY-SYSTAB` and `CHANGE-SYSTAB`, respectively. You can also display the settings of congestion onset, abatement, and discard tables by using the SINAP/SS7 system utility, `sy`, from any SINAP/SS7 login window. These processes are described in Chapter 4 of the *SINAP/SS7 User's Guide* (R8051).

Implementing Multiple Link Congestion Functionality

To implement multiple link congestion functionality in the CCITT or China network variant, you must set the environment variable for this function **before** starting the SINAP/SS7 software. You can do this when you initially set up the network, or you can stop the system and return to the UNIX prompt. Enter a value for the link congestion environment variable and restart the SINAP/SS7 system. The system then implements congestion handling when it is needed according to the values you defined. See Appendix B, ‘‘SINAP/SS7 Environment Variables,” for more information on setting environment variables.

Table 3-18 provides the environment variables available for the CCITT_CONGESTION_OPTION.

Table 3-18. Environment Variables for CCITT and China Link Congestion *(Page 1 of 3)*

Environment Variable	Type of Link-Congestion Handling to Use
INTERNATIONAL_1_CONGESTION	The international signaling network option that provides a single congestion onset threshold (CONON1) and a single congestion abatement threshold (CONAB1). It also uses a single discard threshold (CONDIS1) . If you do not define the CCITT_CONGESTION_OPTION environment variable, the CCITT and China network variants implement the International One Congestion Onset, One Congestion Abatement option as the default method of handling link congestion.
NAT_MUL_CONG_WITH_PRIO	The national signaling network option that allows multiple signaling link congestion levels with congestion priority. This option is available to both the CCITT and China network variants by setting the CCITT_CONGESTION_OPTION to NAT_MUL_CONG_WITH_PRIO. This option allows client applications to set congestion priority based on multiple congestion levels (0-3). This option uses these thresholds: • Congestion Onset (CONON1, CONON2, CONON3) • Congestion Abatement (CONAB1, CONAB2, CONAB3) • Congestion Discard (CONDIS1, CONDIS2, CONDIS3) **Note:** This option is the default method of handling link congestion for the ANSI, TTC, and NTT network variants.

Table 3-18. Environment Variables for CCITT and China Link Congestion *(Page 2 of 3)*

Environment Variable	Type of Link-Congestion Handling to Use
NAT_MULT_CONG_WO_PRIO	The national signaling network option that allows multiple signaling link congestion levels without congestion priority. *Available in the CCITT and China network variants*, this option allows the SINAP/SS7 system to maintain up to four levels of link congestion (0-3) and to set a link's congestion status according to these levels. If you specify this option, you must also specify values for the following additional environment variables: • CONGESTION_STATUS - Specifies the level of link congestion (0, 1, 2, or 3) that your SS7 network supports. (Level 0 is the lowest; level 3 is the highest.) The value 2 specifies that the SINAP node supports three levels of link congestion (0, 1, and 2). The value 3 specifies that the SINAP node supports all four levels of link congestion (0, 1, 2, and 3). • CONGESTION_INITIAL_VALUE - Defines the initial link congestion level that the SINAP/SS7 system uses to determine the occurrence of congestion on a link. When the number of MSUs on the link's SS7 driver queue exceeds the congestion onset threshold for the link congestion level defined by this environment variable, the SINAP node considers the link to be congested with this level. Valid values are 1, 2, or 3. The default value is set to 1. This value should not be greater than the value of the variable CONGESTION_STATES. • CONGESTION_TX_TIMER - Defines the interval in seconds between congestion onset measurements. The valid range for this value is 1 through 255 seconds (the default is 1). When this timer expires, the SINAP/SS7 system counts the number of messages on the link's SS7 driver queue and if the number of messages exceeds the value of the congestion onset threshold, the SINAP/SS7 system increments the link's congestion status by 1.

Table 3-18. Environment Variables for CCITT and China Link Congestion *(Page 3 of 3)*

Environment Variable	Type of Link-Congestion Handling to Use
NAT_MULT_CONG_WO_PRIO *(Continued)*	• CONGESTION_TY_TIMER - Defines the interval in seconds between congestion abatement measurements. The valid range for this value is 1 through 255 seconds (the default is 1). When this timer expires, the SINAP node counts the number of MSUs on the link's SS7 driver queue. If the number of MSUs on the queue is less than the value of the congestion abatement threshold, the SINAP node decrements the link's congestion status by 1.

Multiple Congestion States Without the Congestion Priority

When a link is not congested, the link's congestion status is 0. The SINAP/SS7 system considers a link congested when the number of messages on the link's SS7 driver queue exceeds the congestion-onset threshold for the congestion level defined by the environment variable, CONGESTION_INITIAL_VALUE. For example, if CONGESTION_INITIAL_VALUE is set to the value 2 (congestion level 2), the SINAP/SS7 system compares the number of messages on the link's queue to the value of the CONON2 table. As long as the number of messages on the queue is less than the value of CONON2, the link is not considered congested. When the number of messages exceeds this threshold, the link is considered congested. For example, if the value of CONON2 is 110, the link becomes congested when the number of messages on its queue exceeds 110. When this happens, the SINAP/SS7 system sets the link's congestion status to the value specified by the command, CONGESTION_INITIAL_VALUE, and starts the timers, CONGESTION_TX_TIMER and CONGESTION_TY_TIMER.

When the CONGESTION_TX_TIMER timer expires, the SINAP/SS7 system measures the number of messages on the link's SS7 driver queue. If the number of messages on the queue exceeds the value of the next higher level's congestion-onset threshold (specified in the appropriate CONON table), the SINAP/SS7 system increments the link's congestion status by 1 and restarts the CONGESTION_TX_TIMER timer. For example, if a link's congestion status is currently 1, and the number of messages on its queue exceeds the value of the CONON2 table, the SINAP/SS7 system increments the link's congestion status from 1 to 2.

When the CONGESTION_TY_TIMER timer expires, the SINAP/SS7 system measures the number of messages on the link's SS7 driver queue. If the number of messages on the queue is less the value of the next lower level's congestion abatement threshold (specified in the appropriate CONAB table), the SINAP/SS7 system decrements the link's congestion status by 1 and restarts the CONGESTION_TY_TIMER timer. For example, if a link's congestion status is currently 3 and the number of messages on its queue is less than the value of the CONAB2 table, the SINAP/SS7 system decrements the link's congestion status from 3 to 2.

The SINAP/SS7 system continues to count messages and restart the congestion timers until the number of messages on the link's SS7 driver queue is less than the congestion abatement

threshold defined by `CONAB1`. When the number of messages on the link's queue drops below this value, the SINAP/SS7 system no longer considers the link congested.

NOTE

The information in the "Measuring Congestion for Multiple Congestion States Without Congestion Priority Option" section is described in Table 3-18.

Notifying the Application of Congestion

The SINAP/SS7 system notifies an application that a link is congested by sending a message to the application's interprocess communications (IPC) queue. The application can then either stop sending messages or reduce the number of messages it sends, until the congestion level returns to normal. The SINAP/SS7 system sends a message after the application has sent eight outgoing messages over a congested link; in addition, the SINAP/SS7 system writes a message to its alarm log.

The message can be either of the following:

- If an application is registered to receive input at the SCCP or TCAP boundary, the SINAP/SS7 system sends the application an `I_N_PCSTATE_INDIC` message.
- If an application is registered to receive input at the MTP boundary, the SINAP/SS7 system sends the application an `I_MTP_STATUS` message.

Link Congestion Thresholds

The thresholds for each link congestion level are defined by separate sets of system tables. Congestion onset (CONON) tables define the upper threshold for a particular link congestion level. Congestion abatement (CONAB) tables define the lower threshold. Congestion discard (CONDIS) tables define the threshold before messages are discarded.

- `CONON1`, `CONAB1`, and `CONDIS1` define the congestion-level-1 thresholds.
- `CONON2`, `CONAB2`, and `CONDIS2` define the congestion-level-2 thresholds.
- `CONON3`, `CONAB3`, and `CONDIS3` define the congestion-level-3 thresholds.

Table 3-19 lists each congestion table and its default value.

Table 3-19. Congestion Thresholds

Threshold	Default Value
CONON1	80
CONAB1	50
CONON2	110
CONAB2	90
CONON3	140
CONAB3	120
CONDIS1	130
CONDIS2	170
CONDIS3	210

NOTE

In the CCITT network variant, if you do not implement the national signaling network option (multiple link-congestion states with or without congestion priority), the SINAP/SS7 system defaults to the international signaling network option, which has only one congestion onset and one abatement level. The SINAP/SS7 system then uses the threshold levels for `CONON1` and `CONAB1` to determine a link's congestion status.

Priority, Sequence Control, and Quality of Service

This section describes several parameters you can use in an application to specify how the SINAP/SS7 system should process outgoing MSUs. Two parameters support priority and sequence control. Two parameters support protocol class and return option and define quality-of-service (QOS) characteristics. The include files for these parameters are located in the `$SINAP_HOME/Include` directory.

The following list describes each parameter and indicates the structure field in which you define its value.

NOTE

For examples of how to code your TTC and NTT network variant's application to use these parameters, see the sample TTC programs `tcsend.c` and `tcrecv.c` in the directory `$SINAP_HOME/Samples/ttc`.

- *Priority* specifies the message priority for the MSU. This parameter is valid only for SCCP class-0 and class-1 messages. Valid values are in the range `0` through `3` (lowest to highest). Table 3-20 shows the structures and fields in which to define this parameter, depending on the application type.

Table 3-20. Priority Parameters' Structure and Field

Application Type	Structure	Field
TCAP	`tc_dhp_t`	`priority`
	`tc_thp_t` (ANSI)	`priority`
SCCP	`sccp_ctrl_t**`	`sccp_msg_priority`
MTP	`msu_t`	`li` †

**For ANSI or China network variants, bits 5 and 6 of the `msu_t` structure's `sio` field are for the message priority (with possible values from 0 through 3). For the CCITT variant, no message priority is defined or used.
† MTP applications for the TTC and NTT network variants define priority in the two high-order bits, 8 and 7, of the `msu_t` structure's `li` field; the remaining bits, 6 through 1, continue to be used to specify length.

- *Sequence control* specifies the value to use for the signaling link selection (SLS) field of the MSU's MTP routing label. This parameter is valid only for SCCP class-1 messages. Valid values are in the range `0` through `15` for TTC and NTT, and `0` through `31` for all other stacks excluding ANSI.

For the ANSI network variant, the default value placed into the sequence control parameter is `0` through `31`. This same value is used in the sequence control parameter when you select a five-bit SLS using the `CHANGE-SLSTYPE` MML command. However, if you select an eight-bit SLS through the `CHANGE-SLSTYPE` command, the value placed in the sequence control parameter is in the range of `0-255`. See "SINAP/SS7 Interaction with the SS7 Network" in Chapter 2 for more details.

The SLS field determines the link over which the MSU is routed. You can route multiple MSUs over the same link by assigning the same SLS value to each MSU. Table 3-21 shows the structure and field in which to define this parameter, depending on the application type.

Table 3-21. Sequence Control Structures and Fields

Application Type	Structure	Field
TCAP	`tc_dhp_t`	`seq_control`
	`trans_id_t` (ANSI)	`sccp_seq_ctrl`
SCCP	`sccp_ctrl_t`	`sccp_seq_control`
MTP	N/A	N/A

- *Protocol class* specifies the type of protocol class to use when sending the MSU, and *return option* specifies what the SINAP/SS7 system should do if an error occurs. You use a single value to define both parameters, as described in Table 3-22.

Table 3-22. Protocol Class and Return option Values

Value	Description
`0`	Connectionless class 0, no return on error
`1`	Connectionless class 1, no return on error
`0x80`	Connectionless class 0, return on error
`0x81`	Connectionless class 1, return on error

Table 3-23 shows the structure and field in which to define the return-option and protocol-class parameters, depending on the application type.

Table 3-23. Return Option and Protocol Class Parameters Structure and Field

Application Type	Structure	Field
TCAP	`tc_dhp_t`	`qlty_of_svc`
	`tc_thp_t` (ANSI)	`qlty_of_svc`
SCCP	`sccp_user_t`	`ret_prot`
MTP	N/A	N/A

MTP User Flow Control

The SINAP/SS7 system supports MTP *user flow control* for the CCITT, China, and ANSI network variants. The TTC and NTT network variants do not support user part unavailable (UPU) messages. (the SINAP/SS7 system's implementation of this feature conforms to Telcordia specifications and follows the standards set forth in ITU-T (CCITT) Recommendation Q.704 (1988 and 1993, Sections 11.2.7 and 15.17) UPU functionality is part of MTP Level 3 signaling traffic flow control. This feature enables MTP to send a UPU message to an origination user part (that is, an application) when the SINAP/SS7 system cannot deliver an incoming message to its destination. The origination application can then arrange to stop sending messages to that destination until it becomes available again.

NOTE

This feature is **required** by the China network variant.

The remainder of this section explains how to implement MTP user flow control and describes how the SINAP/SS7 system generates a UPU message and responds to a UPU message from another point code.

Implementing MTP User Flow Control

To activate the MTP user flow control feature, define the environment variable `MTP_USER_FLOW_CTL` at a UNIX prompt **before** you start the SINAP/SS7 system. (For instructions, see ''Defining SINAP/SS7 Environment Variables'' in Appendix B.)

NOTE

You need not assign a value to the `MTP_USER_FLOW_CTL` environment variable; the SINAP/SS7 system verifies the existence of the variable.

To turn off the MTP user flow control feature, remove the definition for the `MTP_USER_FLOW_CTL` environment variable by following the appropriate procedure for the shell you are using. For example, if you are using the C shell, issue the command: `unsetenv MTP_USER_FLOW_CTL`.

If the `MTP_USER_FLOW_CTL` variable is not defined, the SINAP/SS7 system does **not** generate a UPU message when it cannot deliver an incoming message, even if the destination user part is unavailable. If the variable is defined, the SINAP/SS7 system generates a UPU message when it receives an incoming message that it cannot deliver.

For ISUP messages, no UPU will be sent if ISMG is running, even if no ISUP application is running. To overcome this ISUP limitation, a new feature, controlled by a SINAP environment variable called "`ISUP_UPU_FEATURE`", is introduced. ". When the user sets `ISUP_UPU_FEATURE` to 1 in `sinap_env.[sh|csh]`, the feature will be activated. At the same time, SINAP will implicitly activate the environment variable called

"MTP_USER_FLOW_CTL". If the user wants "MTP_USER_FLOW_CTL" only, please do not specify "ISUP_UPU_FEATURE".

When the feature is activated, SINAP will send User Part Unavailability (UPU) message to each remote point code from incoming ISUP messages when the ISUP user application is down. If both ISMG and SCRs are down, the UPU message will contain the cause of UNEQUIPPED, if ISMG is up but SCR is down, the UPU message will contain the cause of INACCESSIBLE. The UPU message will be only sent once for the same remote point codes. The ANSI variant of SINAP/SS7 is based on T1.111 (1990/1992), which does not have "Unavailability cause" field (added in 1996 T1.111) in the UPU message. As shown at 15.17.2/T1.111.4 (1992), these four bits are coded "0000" as Spare field in UPU. This new feature be automatically disabled for DLPC/LPCR configuration, even if we have the environment variable ISUP_UPU_FEATURE set. This is because in DLPC/LPCR configuration, SINAP will be acting as an STP. Hence, we can send TFP message to the remote ends to stop ISUP traffic flow.

Generating a UPU Message

If you define the MTP_USER_FLOW_CTL environment variable, the SINAP/SS7 system generates a UPU message when either of the following situations occur.

- When an incoming MSU cannot be delivered because the specified destination user part is unavailable, the SINAP/SS7 system's MTP Level 3 (L3) sends a UPU message to the MTP L3 on the OPC and generates the following event, where *%s* is a signaling information octet (SIO) or SSN string and *%d* is an SIO or SSN number.

  ```
  Lost MSU, Process Not Found (%s %d)
  ```

- When the specified user part's input queue becomes full, or when the number of MSUs on the queue has reached a threshold defined by the formula `threshold = Q - (Q / 10)`, where `Q` is the number of MSUs that the input queue holds. For example, if the user part's input queue holds 50 MSUs, the SINAP/SS7 system will generate UPU messages for that user part when there are 45 MSUs on the input queue; if the input queue holds 40 MSUs, its threshold is 36.

The SINAP/SS7 system supports several UPU message unavailability-cause reasons which are documented in the 1993 edition of ITU-T recommendation Q.704, section 15.17. They explain

why MTP could not deliver a message to its destination. Table 3-24 describes the meaning of these messages.

Table 3-24. Unavailability-Cause Values for UPU Messages

Unavailability-Cause Value	Numeric Values	Description
`UPU:unknown`	`0`	Unknown reason
`UPU:unequipped remote user`	`1`	The user part is not equipped.
`UPU:inaccessible remote user`	`2`	The user part is equipped but not accessible.

Handling Incoming UPU Messages

When the SINAP/SS7 system receives an incoming UPU message, it means that one of the applications running on the SINAP/SS7 system sent an outgoing message to a remote node but the message could not be delivered to its destination. On receipt of an incoming UPU message, the SINAP/SS7 system examines the `upu_id_cause` field of the nested structure `m_block_t.ud.ccitt_msu.mtp_ud.upu` to determine why the outgoing message could not be delivered. The `upu_id_cause` field, previously named `user_part_id`, contains either network-congestion information or a UPU unavailability-cause reason, as described in Table 3-25.

Table 3-25 describes the meaning of the bits in the `mtp_status_t` structure's `status` field and the `scmg_ipc_t.primitives.pcstate` structure's `pc_status` field. Bit 7 indicates the field's contents. If bit 7 is `0`, the field's 0 and 1 bits contain network-congestion information. If bit 7 is `1`, the field contains a UPU unavailability-cause reason: bits 4 through 6 contain the reason, and bits 0 through 3 contain the user part ID.

NOTE

The unavailability-cause reason is stored in the field's upper four bits.

Table 3-25. Status Field Bits

Bit Groupings	Bits in the `status` or `pc_status` Field 7	6	5	4	3	2	1	0	Description
Group 1: Congestion Information	0	0	0	0	0	0	0	0	Congestion level 0
	0	0	0	0	0	0	0	1	Congestion level 1
	0	0	0	0	0	0	1	0	Congestion level 2
	0	0	0	0	0	0	1	1	Congestion level 3
Group 2:† UPU Message Information	1	0	0	0	*y*	*y*	*y*	*y*	Unknown
	1	0	0	1	*y*	*y*	*y*	*y*	Unequipped remote user‡
	1	0	1	0	*y*	*y*	*y*	*y*	Inaccessible remote user

† In the UPU message information bits, *yyyy* is the user-part ID, which is formatted according to ITU-T Recommendation Q.704, Section 15.17.4.
‡ If the UPU message information bits indicate unequipped remote user, the SINAP/SS7 system cannot send a primitive to the user part because it is not currently active. In this case, the SINAP/SS7 system logs an event message to the Alarm Log file and discards the UPU message.

The SINAP/SS7 system uses the information in the `upu_id_cause` field to generate an `MTP-STATUS` primitive that it sends to the appropriate MTP user part via IPC. If the `upu_id_cause` field contains a UPU unavailability-cause reason, the SINAP/SS7 system sends an `MTP-STATUS` primitive to the MTP user part identified by bits 0 through 3 (the *user part ID*). For example, on receipt of an incoming UPU message with a user part ID of 3 and an unavailability-cause reason of 0 (unknown), the SINAP/SS7 system sends an `MTP-STATUS` primitive to SCCP management (SCMG) because SCMG's MTP user part ID is 3. SCMG then uses the information in the `MTP-STATUS` primitive to generate an `N-PCSTATE` primitive, which it then broadcasts to all registered SCCP user parts.

NOTE

If the user part ID identifies multiple user parts, as in the case of an application with multiple instances, the SINAP/SS7 system generates an `MTP-STATUS` primitive for each user part. If the

specified user part is not active, the SINAP/SS7 system logs an event message to the Alarm Log file and discards the UPU message.

To evaluate the UPU message's unavailability-cause reason, your application should examine the `MTP-STATUS` or `N-PCSTATE` primitive sent by the SINAP/SS7 system. Use the information in Table 3-25 to interpret the UPU message's meaning. (Note that the structures containing these primitives, `mtp_status_t` and `pcstate`, are defined in the `prims.h` include file.)

- MTP applications should examine the `mtp_status_t` structure, which contains the `MTP-STATUS` primitive. The UPU unavailability-cause reason is defined in the structure's `status` field.
- SCCP applications should examine the `scmg_ipc_t.primitives.pcstate` structure, which contains the `I_N_PCSTATE` primitive. The UPU unavailability-cause reason is defined in the structure's `pc_status` field.

For an example of how to code an application to examine the `MTP-STATUS` primitive, see the `$SINAP_MASTER/Samples/ccitt/mtprecv.c` sample program. For an example of how to examine the `I_N_PCSTATE` primitive, see the `I_N_PCSTATE_INDIC` switch statement in the `$SINAP_MASTER/Samples/ccitt/tcrecv.c` sample program. (`SINAP_MASTER` is the directory in which the SINAP/SS7 software is installed; the default is `/opt/sinap_master` for the HP-UX operating system. You can examine the `MASTER` field in the `/etc/sinap_master` file to determine where the SINAP/SS7 software is installed on your system.)

XUDT and XUDTS Messages (CCITT and China)

SINAP/SS7 networks configured for CCITT or China support the following two types of messages that might be exchanged by applications running in your network:

- *Extended unitdata (XUDT) messages* carry segmented message data for connectionless protocol classes, 0 and 1. The message sends data with or without optional parameters. XUDT message segments are those message signaling unit (message) segment sizes that are smaller than the default or maximum segment size used to exchange messages in the SINAP/SS7 system.
- The SINAP/SS7 system sends *extended unitdata service (XUDTS) messages* to the originating SCCP application when an XUDT message with optional parameters cannot be delivered to its destination because of an error. An XUDTS message is sent only when the `return on error` option is set in the XUDT message.

Applications can exchange XUDT messages if they register with CASL at the SCCP XUDT or TCAP XUDT boundary. However, applications can still use the existing API to exchange unitdata (UDT) messages even if they are registered at the SCCP XUDT or TCAP XUDT boundary.

The mechanism for determining whether the message type is UDT or XUDT differs for applications registering at the SCCP and TCAP boundaries. An application registering at the SCCP boundary specifically builds a message of the type, UDT or XUDT, and sets the desired message size. An application registering at the TCAP boundary uses the total size of all components to decide the message type. The program can override this decision by using the `TC_REQUESTX` primitive in the `tblock` when the `ca_put_tc()` function is called.

The maximum segment size, including the length of the data and address parameter fields in the MSU, depends on the network variant being used.

XUDT functionality segments messages up to 2048 bytes long into multiple XUDT message segments up to a maximum of 16. The maximum segment size (including the length of the data and address parameter fields in the MSU) depends on the network variant you are using. For the CCITT variant, the maximum segment size is 254 bytes. For the China variant, the maximum segment size is 251 bytes.

All XUDT messages that are segments of the same message are assigned the same unique identification number or local reference number (LRN). Each time an LRN is released, it cannot be reused on a node-wide basis for a minimum period of time defined by the SCCP freeze timer, SCTX. This timer defines the time period during which an LRN assigned to multiple segments of the same message is frozen.

NOTE

The SINAP/SS7 system automatically provides the freeze timer value in the current software release and does not implement any changes you might make to the SCCP SCTX timer. See Chapter 4 of the *SINAP/SS7 User's Guide* (R8051) for information on displaying and changing the XUDT timer values.

The message reassembly process must receive all segments and reassemble the message within the time period specified in the SCCP reassembly timer, SCTY. There is one SCTY timer per node. All segments of an XUDT message must be received and reassembled before the SCTY timer expires. The default value is 1 second. If the message is not reassembled in the allowed time period, the SINAP/SS7 system discards the message and sends an XUDTS message to the originating application if the XUDT message had `return message on error` specified in the protocol class field. This error message indicates that the message was not delivered because of an error. The XUDTS messages contains the first segment of the XUDT message. The SINAP/SS7 system handles XUDTS messages and XUDT messages in the same manner.

You can display the XUDT timer values using the following MML command:

```
DISPLAY-SYSTAB:TABID=SCCPTM,TIMER=SCTY;
```

or use the `send_cm` command to issue the command.

You can change an SCCP XUDT timer value through the Terminal Handler using the MML command, `CHANGE-SYSTAB`. See Chapter 4 of the *SINAP/SS7 User's Guide* (R8051) for more information on displaying and changing XUDT timer values.

XUDT MSU Segment Sizes

The system guarantees that each MSU segment generated for an XUDT message is the same size, except for the last segment. You can use the default segment size for your network variant, or define a smaller size.

Issue the following `sy` command to view the default XUDT MSU segment size for your system:

```
#sta,xudt
```

The SINAP/SS7 system uses the maximum segment size as the default for the network variant being used. For CCITT, the maximum segment size is 254 bytes. For China, the maximum segment size is 251 bytes. Both sizes include the data and address fields of the MSU.

You can define a segment size that is smaller than the maximum (default) size by uncommenting the following environment variable in the SINAP environment file, `$SINAP_HOME/Bin/sinap_env.[csh` or `sh]`, when you start the SINAP/SS7 system:

```
SINAP_XUDT_SEGMENT_SIZE
```

NOTES

1. The SINAP/SS7 system uses the default segment size unless you define the `SINAP_XUDT_SEGMENT_SIZE` environment variable, or if you specify a segment size greater than the maximum allowed for your variant.
2. The segment size relates to the size of the MSU going across the network. The segment size is used to tune the network and is not used to determine when TCAP should use UDT or XUDT messages.

Validating the XUDT Message Segment Size

The `nmnp` process validates the value specified in the `SINAP_XUDT_SEGMENT_SIZE` environment variable shared memory static tables. Each process that registers for either the SCCP or TCAP XUDT boundary, reads the value from shared memory and stores it in the global variable, `CA_XUDT_SEG`. The `ca_put_msu_int` function uses the `CA_XUDT_SEG` global variable to perform segmentation processing.

NOTE

The `ca_put_msu_int` function is an internal function called directly by either `ca_put_msu()` or `ca_put_tc()`. See Chapter 6 for more information on these CASL functions calls.

Programming Considerations for XUDT/XUDTS Messages

Application developers should consider the information in the sections that follow when developing applications that use XUDT messages.

CASL Registration

To implement the XUDT/XUDTS functionality, an application must register with CASL using one of the following `ss7_input_boundary` parameters specified in the `register_reg_t` structure in the `register.h` header file.

- `SS7_INPUT_BOUNDARY_SCCPX`
- `SS7_INPUT_BOUNDARY_TCAPX`

Any application that registers using either of these boundaries requires an additional set of buffers for XUDT message segmentation and reassembly processing. The application must use the `reassembly_count` parameter in the `register_reg_t` structure in `register.h` to define the number of reassembly buffers it requires. The number of buffers defined determines the number of incoming XUDT messages that can be simultaneously reassembled at any point in time. CASL uses one reassembly buffer per incoming XUDT message.

The Node Management process `cl_register()` internal function ensures that multiple instances of the same application register at the same input boundary.

An application can also use the existing API to exchange UDT/UDTS messages, even when it is registered at the SCCP XUDT or TCAP XUDT boundary.

SCCP Applications

After registering with CASL using the parameter `SS7_INPUT_BOUNDARY_SCCPX`, an SCCP application builds the XUDT mandatory and variable parameters (defined in `mblock_t` in the `mblock.h` file), builds the data parameter in a private buffer, and inserts the pointer to data and size in appropriate fields in the `mblock_t` structure. When the application program issues a `ca_put_msu()`, CASL segments the message for transmission on the network. When the application issues a `ca_get_msu()` as the XUDT message is read from the network, CASL blocks the application until the complete message is received, then passes the message pointer in `mblock`.

Before sending an XUDT message, an SCCP application program performs the following processing:

- Puts the data into a user buffer large enough to contain the data.

NOTE

The application does not specify the size of the data field, but places a pointer to the user data buffer and ignores the single octet data length field. As the data is segmented, CASL inserts the data length in the data parameter on each MSU. A single

octet cannot specify the length of a large message (more than the maximum allowed for the variant being used).

- Places a pointer to the user data buffer in [`m_block_t->`]`sc_prim.p_user_data` (in the `sccp_prim_t` structure in `mblock.h`).
- Puts the size of the data field in `[m_block_t->]sc_prim.user_data_size` (in the `sccp_prim_t` structure in `mblock.h`). The application is not required to put the overall message size in the `[m_block_t->]mtp_ctrl.msg_size` field.
- Stores the `SC_CTRL_EXUNITDATA` in `[m_block_t->]sc_ctrl.sccp_ctrl` (in the `sccp_ctrl_t` structure in `mblock.h`).
- Uses the `ccitt_sccp_xuser_t` message format structure (for the CCITT network variant) or the `ansi_sccp_xuser_t` message format structure (for the China network variant) to format the XUDT message.
- Stores the `SC_N_EXUNITDATA` in the

  `[m_block_t->]ud.msu.mtp_ud.sccpx.msg_type` field

  Note that this reference and the reference

  ```
  [m_block_t->]ud.ccitt_msu.mtp_ud.sccpx.msg_type
  ```

  can both be used for standard CCITT messages. However, for standard China messages in the China variant, you should modify the reference as follows:

  ```
  [m_block_t->]ud.ansi_msu.mtp_ud.sccpx.msg_type
  ```

- Stores the maximum number of global title translations (up to 15) a message can undergo in the `[m_block_t->]ud.msu.mtp_ud.sccpx.hop_counter` field.
- Stores 0 in the `[m_block_t->]ud.msu.mtp_ud.sccpx.op_off` field, since CASL uses this field internally.
- Initializes the remaining fields the same way it does a for a UDT request.

  NOTE

  All fields are accessed by `[m_block_t->]ud.msu.mtp_ud.sccpx ret_prot, cld_off`, `clg_off`, and `tcap_off`.

- Stores the address fields in the `[m_block_t->]ud.msu.mtp_ud.sccpx.ud` field.

To send an XUDT message an SCCP application calls the `ca_put_msu()` function and passes a pointer to the `m_block_t` structure that holds the XUDT message. CASL then segments the data buffer and sends the number of XUDT MSUs required to carry the data.

To read an XUDT message, the application program calls the `ca_get_msu()` function. CASL blocks the application program until one of the following conditions occurs:

- A complete XUDT message is reassembled. CASL returns a pointer to the `m_block_t` that holds the message. The `[m_block_t->]sc_ctrl.sccp_ctrl` of the message is set to `SC_CTRL_EXUNITDATA`. The data parameter portion of the message is contained in a separate buffer pointed to by `[m_block_t->]sc_prim.p_user_data`. The length of the data parameter is stored in `[m_block_t->]sc_prim.user_data_size`. The mandatory and variable fields are stored in `[m_block_t->]ud.msu.mtp_ud.sccpx.ud`.

  The SCCP portion of the message uses the XUDT format defined in one of the following structures, depending on the network variant being used:

  - `ccitt_sccp_xuser_t`
  - `ansi_sccp_xuser_t` (for the China variant)

- A nonxUDT MSU (`SC_CTRL_UNITDATA`, or `SC_CTRL_NOTICE`) is received. CASL returns a pointer to the `m_block_t` that holds the message. The `[m_block_t->]sc_ctrl.sccp_ctrl` is set to either `SC_CTRL_UNITDATA` or `SC_CTRL_NOTICE`.

  NOTE

  Both UDTS and XUDTS messages are received at the SCCP boundary as `SC_CTRL_NOTICE` messages. It is up to the application to check the `msg_type` field in the SCCP user structure to determine if the message is a UDTS or XUDTS message.

  The data is stored in the `[m_block_t->]ud.msu.mtp_ud.sccp.ud` for `SC_CTRL_UNITDATA` and `SC_CTRL_NOTICE` messages carrying a UDTS. It is stored in `[m_block_t->]ud.msu.mtp_ud.sccpx.ud` for `SC_CTRL_NOTICE` messages carrying a XUDTS. The length of the overall message is stored in `[m_block_t->]mtp_ctrl.msg_size` for both `SC_CTRL_UNITDATA` and `SC_CTRL_NOTICE` messages (in the `mtp_ctrl_t` structure in `mblock.h`).

- There is no data to be returned to the application. In this case, the application, returns `-1` and `errno` is set to `EINTR`. This condition occurs when a blocked read (`ca_get_msu_(wait)`) has been specified. Otherwise, the application returns the error `CA_ERR_NO_MSUS`. In either case, the application should call `ca_get_msu()` again.

When CASL detects a reassembly error such as an out-of-order segment, a reassembly timeout, or a lack of reassembly buffers, it terminates the reassembly processing associated with the message and returns an XUDTS message across the network containing the appropriate error message (assuming the return message on error option was set for the XUDT message). CASL

then continues to process messages in the receive buffer until one of the following conditions occurs:

- A complete XUDT message is reassembled
- A nonxUDT message is received
- Processing of the receive buffer is complete and no messages exist to return to the application

Then, CASL returns the appropriate value.

TCAP Applications

After a TCAP application has registered with CASL using the parameter `SS7_INPUT_BOUNDARY_TCAPX`, CASL allocates 2048-byte component buffers, one per `tblock`. The `[t_block_t->].chp.extnd_data_ptr` in the associated `tblock` (in the `t_block_t` and `tc_chp_t` structures in `tblock.h`) points to each allocated buffer. CASL allocates these component buffers in addition to the XUDT reassembly buffers it allocates. The data size is put in `[t_block_t->].chp.extnd_data_size`.

NOTE

The `tblock` data length field (`[t_block_t->].chp.tot_data_len`) is set to `0`.

If the size of the component fits in a single MSU (less than or equal to `MAX_DATA_SIZE_C`), the component data is put in `[t_tblock_t->].chp.data` and the length is put in `[t_tblock_t->].chp.tot_data_len`.

NOTE

Be sure to set the unused fields of the extended data length field (`[t_block_t->].chp.extnd_data_size` and/or `[t_tblock_t->].chp.tot_data_len)` to `0`.

Once the reassembly and component buffers are allocated, the TCAP application can build large messages using XUDT segmented messages (greater than 240 octets), and/or put many segments (as many as fit in a 2048-byte buffer). The application issues a `ca_put_tc()` function call with the `tblock` index as a parameter. As the application issues each `ca_put_tc()`, it saves the message segments in a buffer until the application issues a `ca_put_tc()` for the transaction. Then the TCAP application builds all segments in a buffer and calls one of its internal functions (`ca_put_msu_int()`) to write the message to the network.

CASL segments the message and sends the number of MSUs required to move the message across the network. At this point CASL determines whether to use a UDT or XUDT message to carry the data based on the total length of the combined components. CASL sends a UDT if all

the components fit into a single MSU. Otherwise, CASL sends an XUDT message if the combined size of the components is less than or equal to 2048 bytes.

The application can ensure the components are carried in a XUDT message, by using the `TC_REQUESTX` primitive in the `[t_block_t->].primitive_type` field of the transaction component. The use of the `TC_REQUESTX` primitive is only valid for applications registered at the `SS7_INPUT_BOUNDARY_TCAPX` input boundary).

The XUDT message also contains a mandatory hop counter parameter which limits the number of global title translations (GTTs) that can be performed on the message. On outbound TCAP messages, (TCAP) CASL provides a default count of 12. You can override this default value by inserting a value in `[t_tblock_t].dhp.hop_count` (in the `tc_dhp_t` structure in `tblock.h`).

You can set an environment variable (`TCRELAY=X`) and the SINAP node will update the `hop_count` field in the `tc_dhp_t` message (in the `t_block_t` structure) with the incoming `hop_count` value. This allows TCAP applications to access and process the latest hop counter values.

Valid range for the hop counter is 1 through 15. CASL uses the value (if specified) in `dhp.hop_count` to write the TCAP message.

When the application issues a `ca_get_tc()` function call (either blocked or unblocked) and the message is contained in an XUDT message, CASL blocks the application until the complete message is received. The data portion of the message is parsed for components. The `ca_get_tc()` returns an index to the `tblock` or an error (the same as for UDT messages). If the component fits in the `tblock`, it is returned in the `tblock`. If the component is larger than the data area of the `tblock`, a new pointer (`chp.extnd_data_ptr`) and size field (`chp.extnd_data_size`) point to the large component.

XUDT Message Formats

The XUDT message format is different from the UDT message format, as shown in Table 3-26. Your application should use the structure in `mblock.h` for the message format that corresponds to the network variant you are using.

Table 3-26. XUDT Message Format

Network Variant	Structure to Use
CCITT	`ccitt_sccp_xuser_t`
China	`ansi_sccp_xuser_t`

NOTE

Since the China network variant uses ANSI-style point code and message formats, the China variant must use the ANSI structure for XUDT message formats.

1996 ITU-T SCCP XUDT/XUDTS Importance Parameter Support

To enable 1996 ITU-T SCCP Importance Parameter (1996 ITU-T Q.713 3.19) support in XUDT/XUDTS messages (CCITT only), define the following environment variable before starting the SINAP/SS7 system:

```
SCCP_ITU96_IMPORTANCE_PARM
```

For user application registering at SS7_INPUT_BOUNDARY_SCCPX boundary, a field - importance_parm - has been added at mblock.h sccp_ctrl_t data structure as U8 data type to represent for the 3-bit Importance values. Since 0 is also a valid Importance value, the MSB of importance_parm is used as a flag to indicate if Importance parameter is included and the 3 LSBs of importance_parm arethe Importance values.

For user application registering at SINAP SS7_INPUT_BOUNDARY_TCAPX boundary, a field - importance_parm - has been added at tblock.h tc_dhp_t data structure, which has the similar representation/usage as the importance_parm at mblock.h sccp_ctrl_t.

To access or set the values of the Importance Parameter at the XUDT message received or to be sent, the user application must access or set the importance_parm field at the corresponding m_block_t or t_block_t data structure accordingly.

Processing SCCP Subsystem Tests in XUDT Messages

To enable SCCP subsystem tests in XUDT messages (CCITT only), define the following environment variable before starting the SINAP/SS7 system:

```
CCITT_XUDT_SCMG
```

If you do not define this variable, the SINAP/SS7 system discards any XUDT SCCP management (SCMG) messages it receives since SCMG messaging is normally handled only by UDT messages.

If the `CCITT_XUDT_SCMG` environment variable is enabled, then the SINAP node: sends a subsystem allowed (SSA) message to the original calling address if the subsystem (SSN) specified in the XUDT message is allowed. The SSA is sent in a UDT message.

You do not need to specify a value for the environment variable. The SINAP/SS7 system only verifies the existence of the variable.

To enable SCCP subsystem tests in XUDT messages automatically each time you start the SINAP/SS7 system, uncomment the `CCITT_XUDT_SCMG` environment variable in `$SINAP_HOME/Bin/sinap_env.[csh or sh]`.

Handling SNM Messages with Nonzero SLCs

SINAP/SS7 networks configured for CCITT, China, or ANSI support handling SNM messages with non-zero SLCs. The 1988 ITU-T (CCITT) MTP standards (Q.704) require that all MTP Level 3 SNM messages use a signaling link code (SLC) of 0. One feature of the 1993 ITU-T (CCITT) (Q.704) standards is the ability to use a nonzero SLC value for any SNM message that is not related to a signaling link, such as a signaling route management (SRM) message.

In the CCITT and China network variants, you must activate this feature on a CCITT or China network variant node by defining the environment variable `MTP_WHITE_BOOK_SLC` on that node **before** starting the SINAP/SS7 system. You need not assign a value to the variable. The SINAP/SS7 system simply verifies that the variable exists. For the ANSI network variant, you do not need to specifically set this variable; non-zero SLC is the default. For instructions on defining variables, see Appendix B, ‘‘SINAP/SS7 Environment Variables.”

If the `MTP_WHITE_BOOK_SLC` variable is not defined, the SINAP/SS7 system discards any incoming SNM messages whose SLC is not 0. If the variable is defined, the SINAP/SS7 system allows incoming SNM messages to have an SLC value other than 0.

NOTE

In the ANSI network variant, there is no need to set the `MTP_WHITE_BOOK_SLC` variable. The ANSI variant allows nonzero SLC as the default behavior.

The MTP Restart Process

The Message Transfer Part (MTP) restart process enables the MTP at a signaling point that has just become available to bring sufficient signaling links into the *available* state to handle expected traffic and to stabilize its routing before user traffic is restarted to the signaling point. The MTP restart process helps prevent routing problems that can occur after the system resumes sending user traffic following a failure due to invalid routing information or too many parallel activities, such as link activation or changeback. MTP restart ensures that the system has sufficient signaling links available to handle the expected traffic volume and the route to the signaling point is stable. A signaling point is *unavailable* if all connected links are unavailable. It becomes *available* when at least one link connected to the signaling point becomes available.

MTP restart is supported by the CCITT, ANSI, and China network variants and adheres to the 1993 ITU-T recommendations for MTP and the 1992 ANSI standards for MTP. The TTC and NTT network variants do not support MTP restart functionality.

Since MTP restart requires links to other nodes in the SS7 network, MTP restart functionality does not apply to the operation of a single node. If, for any reason (such as testing or performing local loopback procedures), you operate a single SINAP node, you should not enable the MTP restart feature.

You activate the MTP restart process by enabling specific environment variables for the network variant you are configuring. This section describes the MTP restart process and discusses the following topics:

- An overview of MTP processing
- Enabling MTP restart functionality

See the *SINAP/SS7 User's Guide* (R8051) for information about displaying MTP restart information and changing system table timer and link congestion threshold settings.

MTP Restart Processing Overview

If the MTP restart environment variable is set, the SINAP/SS7 system performs MTP restart (also called signaling point restart control (SPRC)) whenever you activate the SINAP/SS7 system on a node. The MTP restart procedure can be applied when a SINAP node is the restarting signaling point or when a node adjacent to the SINAP node is the restarting signaling point. The process provides time for the node's links and routes to come into service before the node begins sending user traffic over them. Throughout MTP restart, the node activates and unblocks its links using normal signaling link management (SLM) procedures. This ensures a smooth flow of traffic through the network.

During MTP restart, the node does not pass user traffic between its applications and the applications running on other nodes. Instead, the node exchanges network-status information with its adjacent nodes using the following types of messages:

- Signaling link test management (SLTM)
- Signaling link test acknowledgment (SLTA)
- Traffic restart allowed (TRA)
- Transfer prohibited (TFP)
- Transfer restricted (TFR)
- Transfer allowed (TFA)
- Transfer restart waiting (TRW) (ANSI only)

These messages indicate the availability of the links and routes between the nodes. Once MTP restart ends, the MTP informs each of its user parts (that is, applications) that they can begin passing user traffic.

Several timers define time limits for MTP restart activities. The timers and values differ between the network variants and are described in the following sections. To change these timer values, see Chapter 4 of the *SINAP/SS7 User's Guide* (R8051).

Enabling MTP Restart Functionality

The SINAP/SS7 system performs the following major steps to execute MTP restart:

- Initiates MTP restart processing
- Processes messages during MTP restart
- Completes MTP restart

The CCITT, China, and ANSI network variants support MTP restart functionality, but are activated by different environment variables and differ slightly in their timer values and functionality. The major differences between the variants are as follows:

- The ANSI network variant has a larger set of timers and protocols concerning messages received, timer expiration, and timer stoppage
- The ANSI network variant contains the traffic restart waiting (TRW) message.

The following sections describe how to run MTP restart on each network variant.

MTP Restart Processing (CCITT and China)

To implement the MTP restart on your system, define the appropriate environment variable on each node, as follows.

- Define the following variable on the node before starting the SINAP/SS7 system: `MTP_WHITE_BOOK_RESTART`. You need not assign a value to the variable. The SINAP/SS7 system simply verifies the variable exists.

  If the variable is defined, the MTP restart process is performed whenever you start or restart the SINAP/SS7 system on the node. If the `MTP_WHITE_BOOK_RESTART` variable is not defined, MTP restart is **not** performed when the SINAP/SS7 system is started or restarted on the node.

- To enable MTP restart functionality automatically each time you start or restart the SINAP/SS7 system, uncomment the `MTP_WHITE_BOOK_RESTART` environment variable in `$SINAP_HOME/Bin/sinap_env.[csh or sh]`.

At the beginning of the MTP restart process, the SINAP/SS7 system activates the `L3T20` timer and marks all concerned routes `ALLOWED`. The `L3T20` timer defines the maximum amount of time allowed to complete all MTP restart activities. The `L3T21` timer defines the amount of time to stop sending traffic to another node because that node is performing MTP restart. (See "Displaying the MTP and SCCP System Tables" in Chapter 4 of the *SINAP/SS7 User's Guide* (R8051) for information about these timers.)

While the MTP restart process is active, the SINAP/SS7 node and its adjacent nodes monitor all of the TFA, TFP, and TFR messages they exchange. The SINAP/SS7 system and its adjacent nodes use the information in these messages to mark those routes available or unavailable and to update the MTP routing tables. The MTP restart procedure works effectively only if the status of the links and routes remains fairly stable.

During MTP restart, the SINAP/SS7 system handles incoming and outgoing messages as follows:

- The SINAP/SS7 system performs normal processing of SLTM and SLTA messages, which have a service indicator (SI) of `0001`. (The SI consists of bits 0 through 3 of the service information octet (SIO) field.)
- The SINAP/SS7 system only processes signaling network management (SNM) message types: TRA, TFP, TFR, and TFA. The SINAP/SS7 system discards all other SNM messages types, such as:
  - Changeover and changeback message (CHM)
  - Signaling-data-link-connection-order message (DLM)
  - Emergency-changeover message (ECM)
  - Signaling-traffic-flow-control message (FCM)
  - Management inhibit message (MIM)
  - Signaling-route-set-test message (RSM)
  - User part flow control (UFC) message groups

  SNM messages have a service indicator (SI) of `0000`.
- The SINAP/SS7 system discards all incoming and outgoing MTP messages whose SI is not `0000` or `0001`.
- The SINAP/SS7 system discards all incoming and outgoing messages (user traffic) for all types of applications: TCAP, SCCP, and ISUP.

Completing MTP Restart

For the CCITT, China, and ANSI network variants, MTP restart ends when the designated timers expire, or when the SINAP/SS7 node receives TRA messages over more than half of all currently activated link sets, (whichever occurs first).

At the completion of MTP restart, the SINAP/SS7 node sends a TRA message to each of its adjacent nodes to indicate it is ready to accept user traffic. The SINAP/SS7 system informs each of its local MTP user parts that restart has ended by sending each user part an `MTP-RESUME` primitive that indicates the accessibility of each adjacent node. The SINAP/SS7 node can then resume passing user traffic for its MTP user parts.

NOTE

If MTP restart is enabled, the SINAP/SS7 system does not send the `MTP-RESUME` primitive a specific timer (`L3T20` for CCITT/China variants or `L3T23` for ANSI variants) expires or until MTP restart ends. If MTP restart is not enabled, the

SINAP/SS7 system sends a primitive whenever a link set or route set comes into service.

See "Displaying the MTP and SCCP System Tables" in Chapter 4 of the *SINAP/SS7 User's Guide* (R8051) for a description of the MTP restart timers and valid timer settings.

MTP Restart Processing for the ANSI Network Variant

To enable the SINAP/SS7 system to perform MTP restart (based on the 1992 edition of ANSI standards) on a particular node, define the following environment variable before starting the SINAP/SS7 system:

```
MTP_ANSI92_RESTART
```

You need not assign a value to the environment variable. The Node Management Parent (`nmnp`) process validates the existence of the environment variable.

NOTES

1. You should be familiar with the fundamental operation of the ANSI MTP restart process as indicated in Section 9 of the ANSI T1.111.4 standards. MTP restart is not applicable to wholly single node operation. Disable the feature if running a single node since this feature requires links to other nodes in the SS7 network.
2. If you do **not** define this variable, the ANSI restart functionality is not implemented and the network processing is based on the 1990 ANSI standards for MTP, which do not include restart functionality.
3. To enable MTP restart functions automatically each time you start the SINAP/SS7 system, uncomment the `MTP_ANSI92_RESTART` environment variable in `$SINAP_HOME/Bin/sinap_env.[csh or sh]`.

MTP restart can be applied in two different ways: using a SINAP node as the restarting signaling point, **or** using a node adjacent to the SINAP node as the restarting signaling point. Both options are described in the following sections. Only a full restart can be performed; the ANSI network variant does not support a partial restart.

ANSI MTP Level 3 timers define the maximum amount of time the restart signaling point or its adjacent signaling points can take to perform specific MTP restart tasks. You can access these timers by executing the MML command, `CHANGE-SYSTAB`. (See Chapter 4 of the *SINAP/SS7 User's Guide* (R8051) for more information.)

Message Processing During MTP Restart

The signaling point carries out link activation procedures on as many other unavailable links as possible. When the first link goes into the in-service state at Level 2, the restarting signaling point begins accepting only the SNM message types: TFP, TFR, TFA, TRA, TRW, CBD, and CBA that have an SI of 0000 and SLT and SLTA messages with an SI of 0010. It discards any other SNM message types.

While a SINAP node is restarting, user traffic (for example, SCCP and TCAP traffic) is discarded. When an adjacent node is restarting, user traffic to and from the adjacent destination is discarded.

Performing MTP Restart on a SINAP Node

If the environment variable, `MTP_ANSI92_RESTART` is defined and all links from this SINAP node are unavailable, the node initiates a full MTP restart.

If the environment variable is defined, the interval for which the SINAP node is unavailable is set to persist for at least the time period defined in timer T27 (2 through 5 seconds). This ensures adjacent points are aware the restarting node is unavailable.

The node attempts to bring a predetermined number of links in each of its link sets into the available state. Links that are transmitting or receiving processor outage status units become ineffective since SINAP does not support local processor outage (LPO). In this case, MTP restart operations resume after timer T27 expires. Messages buffered in MTP Level 2 during the period of unavailability on those transmitting or receiving links are discarded unless the messages were buffered for a period less than the interval defined in timer T1. The SINAP node carries out link activation procedures on as many other unavailable links as possible.

When the first link goes into the in-service state at Level 2 or another route becomes available (triggered by the receipt of a TFA, TFR, or TRA message or by the corresponding link set becoming available), the restarting node processes any TFP, TFR, TFA, TRA, TRW, CBD, and CBA messages. The SINAP node starts T22 and T26 timers either when the first signaling link goes into the in-service state at Level 2, or when the first signaling link becomes available at Level 3.

When the restarting node receives a TRW message before user traffic restarts on the link(s) to the signaling point that sent the TRW message, timer T25 starts. Traffic cannot restart on that link set until the restarting node receives a TRA message or the timer expires.

When the first signaling link of a signaling link set is available, the restarting node immediately restarts the MTP message traffic terminating at the far end of the link set. The node also sends a TRW message to the signaling point at the far end of the link set.

When timer T26 expires, the node restarts timer T26 and sends a TRW message to the adjacent signaling points connected by an available link.

Level 3 management stops timer T22 when sufficient links are available to carry the expected signaling traffic. When timer T22 expires or is stopped, the restarting SINAP node starts timer

T23. During the T23 period, the signaling point receives additional TFP, TFR, TRA, and TRW messages. After TRA messages are received from all available links or Level 3 management determines sufficient TRA messages were received and traffic can be handled, timer T23 is stopped.

NOTES

1. The SINAP/SS7 system counts the number of available links. When 50% of the load-sharing links are available, the SINAP/SS7 system stops timer T22 and starts T23. Timer T23 stops when 50% of the expected TRA messages are received.
2. Although ANSI does not support partial restart, you can simulate a partial restart by assigning a small value to T22 and T23 timers.

When timer T23 stops or expires, timer T26 stops. The restarting node sends TRA messages to adjacent signaling points and starts user traffic by sending users `MTP-RESUME` primitives for all accessible destinations. The node also starts timer T29 for all signaling points to which a TRA message was sent.

If the first link in a previously unavailable link set becomes available while T23 or T24 is running, the restarting node sends a TRW message to the point at the far end of the link.

The SINAP/SS7 system does not support ANSI timers T24 and T30, which are timers for an STP.

Performing MTP Restart on an Adjacent Node

The SINAP/SS7 system considers the MTP of an adjacent signaling point is restarting when the first link in a direct link set is in the in-service state at Level 2 or another route becomes available (triggered by the receipt of a TFA, TFR, or TRA message or by the corresponding link set becoming available). At this point, the SINAP node processes TRW, TRA, TFP, TFR, and TFA messages from the restarting signaling point. Timer T28 starts at this point or when the first signaling link becomes available at MTP Level 3. The changeback procedure is executed at the end of adjacent MTP restart.

If the SINAP node receives a TRW message from the adjacent restarting signaling point while timer T28 is running or before T28 starts, the node starts timer T25 and stops T28 if it is running. If the node receives a TRW message from the adjacent restarting signaling point while timer T25 is running, the node restarts timer T25.

When the first link in a link set to the adjacent restarting point becomes available, the SINAP node sends a TRA message to the adjacent restarting signaling point.

After the SINAP node receives a TRA message from the adjacent restarting point, the SINAP node stops T25 or T28 (whichever is running) and restarts traffic on the link set to the adjacent

restarting point. The node sends `MTP-RESUME` primitives to users concerning the adjacent restarting point and any destinations made accessible by it.

When timer T28 expires, the SINAP node restarts traffic on the link set to the adjacent restarting point if a TRA message was not sent to it. If one was sent, the node starts T25, completes sending TFP and TFR messages, and sends a TRA message. Then, unless a TRW message was received from the adjacent restarting point without a subsequent TRA message, the SINAP node stops timer T25 and restarts traffic on the link set to the adjacent restarting point.

If timer T25 expires, the node restarts traffic on the link set to the adjacent restarting signaling point. After traffic restarts, if the node does not receive a TRA message from the adjacent restarting point, the node restarts timer T29. If the node receives an unexpected TRA or TRW message from an adjacent restarting signaling point, the node returns a TRA message to the adjacent restarting point originating the unexpected message and starts timer T29.

A received TRW or TRA message is not unexpected if T22 or T23 is running and a direct link is in-service at Level 2 to the point originating the message, or if T25, T28, T29, or T30 is running for the signaling point that originated the message.

Completing MTP Restart

For the CCITT, China, and ANSI network variants, MTP restart ends when the designated timers expire, or when the SINAP/SS7 node receives TRA messages over more than half of all currently activated link sets, (whichever occurs first).

At the completion of MTP restart, the SINAP/SS7 node sends a TRA message to each of its adjacent nodes to indicate it is ready to accept user traffic. The SINAP/SS7 system informs each of its local MTP user parts that restart has ended by sending each user part an `MTP-RESUME` primitive that indicates the accessibility of each adjacent node. The SINAP/SS7 node can then resume passing user traffic for its MTP user parts.

NOTE

If MTP restart is enabled, the SINAP/SS7 system does not send the `MTP-RESUME` primitive until timer `L3T20` (CCITT/China) or `L3T23` (ANSI) expires or until MTP restart ends. If MTP restart is not enabled, the SINAP/SS7 system sends a primitive whenever a link set or route set comes into service.

See “Displaying the MTP and SCCP System Tables” in Chapter 4 of the *SINAP/SS7 User’s Guide* (R8051) for a description of the MTP restart timers and the valid timer settings.

MTP Time-Controlled Changeover

The SINAP/SS7 system supports time-controlled changeover (TCCO) procedures, as defined in the 1993 ITU-T (CCITT) Recommendations and 1992 ANSI Recommendations for MTP. You activate this TCCO feature on a SINAP node by setting the appropriate environment

variable for the network variant being used. TCCO supports handling of long-term or short-term processor outages or changeover orders received from the remote end during the MTP Level 3 T1 timer period.

This section provides an overview of TCCO processing for both short- and long-term outages and describes the environment variable you need to define before starting TCCO on your system.

Overview of MTP TCCO Processing

The SINAP/SS7 system implements TCCO procedures and starts the MTP T1 timer under the following conditions:

- When the SINAP/SS7 system receives a remote processor outage (either short- or long-term) on a link at the remote end and it is not possible to exchange changeover messages because doing so might cause a link failure.
- No signaling path exists between the two ends of the unavailable link so the exchange of changeover messages is impossible.
- A signaling link currently carrying traffic was marked *inhibited* either locally or remotely.

In each case, the Level 3 changeover control (TCOC) function receives the message, `signaling link unavailable`, from the link availability control (TLAC) function. When TCOC receives this message, the SINAP/SS7 system initiates changeover or TCCO activities.

Short-Term Processor Outage

A short-term processor outage is one that terminates **before** the MTP T1 timer expires. If the SINAP/SS7 system receives a changeover order for the unavailable link from the remote end during the T1 timer period, the system initiates normal changeover procedures, completes TCCO procedures, and sends a changeover acknowledgment (COA) to the remote end.

Long-Term Processor Outage

A long-term remote processor outage occurs when the MTP Level 3 TCCO T1 timer expires. To avoid sending old messages when the remote processor outage state terminates, the SINAP/SS7 system discards MTP Level 2 messages in the retransmission buffer and synchronizes sequence numbers.

NOTE

The SINAP/SS7 system flushes old messages and synchronizes MTP Level 2 sequence numbers by failing the link when T1 expires (a long-term processor outage condition). This puts the link out of service, flushes old messages, synchronizes sequence numbers, and starts the initial alignment procedure

If a changeover order is received for the unavailable link after the T1 timer expires, the concerned signaling point responds with an emergency changeover acknowledgment (ECA).

Implementing the TCCO Feature

You must define the appropriate environment variable to enable TCCO functionality for the CCITT, China, and ANSI network variants. TCCO is automatically enabled in the TTC and NTT network variants.

For the CCITT and China network variants, define the following environment variable to implement TCCO functionality based on 1993 ITU-T recommendations for MTP.

```
MTP_WHITE_BOOK_TCCO
```

If you do not define it, the system defaults to TCCO procedures based on the 1988 recommendations.

For the ANSI network variant, define the following environment variable to implement TCCO functionality based on the 1992 ANSI standards for MTP.

```
MTP_ANSI92_TCCO
```

If you do not define it, the system defaults to TCCO procedures based on the 1990 standards.

To implement TCCO features each time you start an ANSI-configured SINAP/SS7 system, add the variable to the `$SINAP_HOME/Bin/sinap_env. [csh or sh]` file.

MTP Time-Controlled Diversion

The SINAP/SS7 system implements a time-controlled diversion (TCD) process when the signaling point at the far end of the link made available is currently inaccessible from the signaling point initiating the changeback order. The SINAP/SS7 system also performs TCD when the concerned signaling point is accessible, but there is no signaling route to it using the same outgoing signaling link(s) or one of the same signaling links from which traffic is diverted.

TCD is primarily used at the end of MTP restart when an adjacent signaling point becomes available. TCD is intended to delay changeback to avoid missequencing messages to destination points after a remote point code restarts. Traffic diversion can be performed at the discretion of the signaling point initiating changeback, as follows:

- On a destination basis for each traffic flow
- On an alternative signaling link basis for all destinations previously diverted on the alternative signaling link
- Simultaneously for a number of alternative signaling links or for all alternative signaling links

A signaling point can also apply TCD for changeback between different link sets, instead of using the sequence control procedure, to avoid possible message missequencing or problems with multiple, parallel changebacks.

When changeback is initiated after MTP restart, the adjacent signaling point stops traffic to the point where it is restarting and diverts traffic to alternative links for an interval defined by timer T3. After that interval, the adjacent signaling point starts traffic on the links made available. The time delay minimizes the probability of out-of-sequence message delivery to the destination point(s).

Implementing TCD Feature for ANSI Network Variant

To enable TCD based on 1992 ANSI standards, define the following environment variable before starting the SINAP/SS7 system:

```
MTP_ANSI92_TCD
```

If you do not define this variable, the system defaults to TCD functionality based on the 1990 ANSI standards. To have the TCD feature defined each time you start the SINAP/SS7 system, add the variable to the `$SINAP_HOME/Bin/sinap_env.[csh or sh]` file.

If the TCD feature is enabled via the environment variable `MTP_ANSI92_TCD`, then the SINAP/SS7 system delays changeback completion 500 ms (the default value of the T3 timer) for each link coming into service. Do not enable TCD if you do not want this delay to occur.

Do not enable the TCD feature via the environment variable `MTP_ANSI92_TCD` if the `MTP_ANSI92_RESTART` environment variable is already set. The restart procedure defined by the `MTP_ANSI92_RESTART` environment variable automatically activates TCD.

NOTE

The Node Management node parent (`nmnp`) process validates the existence of the value specified in this environment variable and stores the value in shared memory in the static tables.

Implementing the MTP Management Inhibit Feature (ANSI)

To enable MTP Management Inhibit based on 1992 ANSI standards, define the following environment variable before starting the SINAP/SS7 system:

```
MTP_ANSI92_MANAGEMENT_INHIBIT
```

If you do not define this variable, the system defaults to MTP Management Inhibit functionality based on the 1988 ANSI standards. To have MTP Management Inhibit based on 1992 ANSI standards defined each time you start the SINAP/SS7 system, add the variable to the `$SINAP_HOME/Bin/sinap_env.[csh or sh]` file.

If the MTP Management Inhibit feature is enabled via the environment variable `MTP_ANSI92_MANAGEMENT_INHIBIT`, then a Link Uninhibit Acknowledgment (LUA) is sent and traffic restarted when no response is received for Link Force Uninhibited (LFU) requests. If, for any reason, an uninhibit signaling link message is not received in response to a link forced uninhibit message, the SINAP/SS7 system waits until timer T13 expires. If this is the first expiry of T13 for this uninhibition attempt on this link, the procedure is restarted including inspection of the status of the inhibited link. If the link is marked failed, blocked, or timer T13 has expired for the second time during uninhibition of this link, management is informed and the uninhibition is abandoned.

Signaling Link Selection (SLS) Message Distribution

The SINAP/SS7 system supports the following types of load distribution:

- *Round-robin distribution* places MSUs sequentially on each of an application's incoming queues
- *Least-utilized distribution* places MSUs on the application's incoming queue with the fewest MSUs.
- *Signaling link selection (SLS)* distributes MSUs based on the value of their SLS field. SLS message distribution ensures that MSUs routed over the same link are handled by the same application instance. It is useful for circuit-related signaling protocols such as Telephone User Part (TUP), which require that MSU sequences be preserved on a per-circuit basis.

When an application instance registers, the SINAP/SS7 system assigns it an SLS code. The SINAP/SS7 system keeps track of the SLS codes assigned to application instances by maintaining a listing of SLS codes for the application. You can display this listing by issuing the appropriate `sy` command (`#SLD`). For instructions, see the section "Displaying SLS Assignments" later in this chapter.

NOTE

When the number of active application instances changes, the SINAP/SS7 system updates its SLS map, which may disrupt an existing dialogue and cause MSUs to be lost. Also, when the number of active links changes, the sender will typically reassign SLS codes, which may necessitate restarting a dialogue that is already in process.

Implementing SLS Message Distribution

To implement SLS message distribution, an application must register with the SINAP/SS7 system as follows:

- The application must register to receive input at the SCCP boundary, or it must register with a service information octet (SIO) instead of an SSN.

- The application must register with the `inbound_load_dist_type` field of the `register_req_t` structure set to `3`, which specifies `SLS_DISTRIBUTION`.

For more information, see the descriptions of the `register_req_t` structure's `sio_ssn_ind`, `sio_ssn`, and `ss7_input_boundary` fields in the description of the `ca_register()` function in Chapter 6, ''CASL Function Calls."

Displaying SLS Assignments

You can display information about an application's load distribution (round-robin, least-utilized, or SLS distribution) by issuing the `#SLD` command through the SINAP/SS7 utility (`sy`). To invoke the utility, type `sy` from the command line of a SINAP/SS7 login window (that is, any window through which you have logged in as the user `SINAP`).

The first line of the `#SLD` command output identifies the application and the number of active application instances. The second line (`SLS Distribution`) displays the SLS code assigned to each of the application's instances. Each position in this line corresponds to an SLS code (`0` through `15`); the number of the application instance assigned this SLS code appears in this position. For example, an SLS distribution of `5, 5, 4, 3...` indicates that application instance `5` is assigned SLS codes `0` and `1`; application instance `4` is assigned SLS code `2`; application instance `3` is assigned SLS code `3`; and so on.

NOTE

In the ANSI network variant, if you specified an eight-bit SLS via the `CHANGE-SLSTYPE` MML command, messages for the appropriate application that contain an eight-bit SLS have the three most significant bits masked out because the instance number of the application is contained in the lower four bits of the SLS and can only take on the values `0-15` (the maximum capability). See "SINAP/SS7 Interaction with the SS7 Network" in Chapter 2 for more details.If you use the SLS message distribution feature with eight-bit SLS processing enabled, the remote sender must only set the lower four bits of the SLS. Setting any other bits will exceed the maximum capabilities of this feature and result in unpredictable system performance. Therefore, the sending application and the receiving application (running on the SINAP node) must agree on the instance number protocol and the SLS field must not exceed `15`.

The following command examples are for an application that uses SLS load distribution; however, you can also issue these commands to display the load-distribution information for an application that uses round-robin or least-utilized load distribution. Note that the pound sign (#) is part of the `#SLD` command.

- To display the SLS mapping for an application that registered with an SIO, enter the following form of the command (where *sio_number* is the SIO).

  ```
  #SLD,SIO,sio_number
  ```

  The following command specifies an application that registered with an SIO of 5. The command output indicates that the application has two instances whose SLS assignments are as follows: application instance 2 is assigned SLS codes 0 through 7, and instance 1 is assigned SLS codes 8 through 15.

  ```
  #SLD,SIO,5
  SIO 5: instance count = 2
  SLS Distribution = 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1
  ```

- To display the SLS mapping for an application that registered with an SSN, enter the following form of the command (where *ssn_number* is the SSN).

  ```
  #SLD,SSN,ssn_number
  ```

  The following command specifies an application that registered with an SSN of 254. The command output indicates that the application has five instances whose SLS assignments are as follows: application instance 5 is assigned SLS codes 0, 1, and 4; instance 4 is assigned SLS codes 3, 10, and 11; instance 3 is assigned SLS codes 2, 8, and 9; instance 2 is assigned SLS codes 5, 6, and 7; and instance 1 is assigned SLS codes 12, 13, 14, and 15.

  ```
  #SLD,SSN,254
  SSN 254: instance count = 5
  SLS Distribution = 5 5 3 4 5 2 2 2 3 3 4 4 1 1 1 1
  ```

- In some cases, the SINAP/SS7 system identifies an application by its name rather than its SSN (for example, if the application implements enhanced message distribution and is one of several applications that use the same SSN). To display the SLS assignments of such an application, issue the following command (where *appl_name* is the name of the application).

  ```
  #SLD,APPL,appl_name
  ```

  The following command specifies a load control application whose name is DB12. The command output indicates that the application has three instances whose SLS assignments are as follows: application instance 3 is assigned SLS codes 0, 1, 2, 8, and 9; instance 2 is assigned SLS codes 5, 6, 7, 10, and 11; and instance 1 is assigned SLS codes 3, 4, 12, 13, 14, and 15.

  ```
  #SLD,APPL,DB12
  APPL DB12: instance count = 3
  SLS Distribution = 3 3 3 1 1 2 2 2 3 3 2 2 1 1 1 1
  ```

Enabling Random SLS Generation

SINAP, by default, conforms to the ANSI T1.111.4 standard (1988) and uses a default SLS of zero for the Route Set Congestion Test (RCT) message. However, always sending the RCT message on the same link within the same link set that the TFC was received on always results in the RCT message testing the same network path, which may or may not be congested. If traffic is not evenly distributed this may result in over controlling (which will occur when the RCT message is routed on the path that is more likely to be congested) or under controlling (which will occur when the RCT message is routed on the path that is more likely to not be congested).

Random Link Selection

To smooth out this effect, the SINAP node provides a feature for the user to enable the generation of a random link selection (SLS) for RCT. The random SLS is placed in the SLS field of the outgoing RCT message.

To enable this feature change `$SINAP_HOME/sinap_env[.sh` or `.csh]`:

```
MTP_RCT_LOAD_SHARING_SLS
```

NOTE

This feature is available for ANSI users only. Setting the environment variable has no effect for other NSP variants.

Setting SLS Bits in the MTP Routing Label

In the TTC and NTT network variants, you can set the SLS bits into the routing label of the MTP, using the macro `NTT_CA_SET_LABEL`.

NOTE

The NTT and TTC network variants both use the macros `TTC_CA_GET_DPC`, `TTC_CA_GET_OPC`, `TTC_CA_GET_SLS`, `TTC_CA_GET_SLC` and `TTC_CA_GET_PRIO` (including `TTC_PRIO_MASK`).

The sample programs in `$SINAP_HOME/samples/ttc` use the macro `CA_SET_LABEL`, which is set to `TTC_CA_SET_LABEL` (by default).

For the NTT variant, the sample programs must use `NTT_CA_SET_LABEL`. For the TTC variant, the sample programs can use `CA_SET_LABEL` or `TTC_CA_SET_LABEL`.

In the ANSI network variant, you can set the SLS bits in the routing label of the MSU using the macro `ANSI_CA_SET_LABEL`. If the SINAP node is configured with the default SLS value (`5`) or a SINAP user sets a five-bit SLS value using the `CHANGE-SLSTYPE` MML command, the SINAP node masks out the upper three most significant bits of the SLS value. However, if a SINAP user selects an eight-bit SLS (via the `CHANGE-SLSTYPE` MML command), the SINAP node uses the full eight bits of the SLS (no masks). See "SINAP/SS7 Interaction with the SS7 Network" in Chapter 2 for more detailed information.

Connection-Oriented Services (CCITT, ANSI, China)

This section describes the SINAP/SS7 connection-oriented feature (COF), which provides SCCP Class 2 and Class 3 connection-oriented services.

- Class 2 provides basic connection-oriented services.
- Class 3 provides connection-oriented services with flow control.

Currently, you can implement connection-oriented services only in SINAP/SS7 configurations that use the CCITT, ANSI, and China network variants for their network protocol, and possibly (although not necessarily) for their TCAP protocol. Since the TTC and NTT standards do not define the use of connection-oriented services, the TTC and NTT variants of the SINAP/SS7 system do **not** support this feature.

This section contains the following topics:

- A processing overview of connection-oriented services
- Information on connection-oriented messages and primitives
- Instructions for activating and implementing connection-oriented services

NOTE

Throughout this section a nested structure (that is, a structure within another structure) is indicated by notation. For example:

`sccp_ipc_t.i_block_t.dest_id`.

This notation refers to the destination field of the `i_block_t` structure, which is contained within the `sccp_ipc_t` structure.

See the *SINAP/SS7 User's Guide* (R8051) for information on administrative considerations for connection-oriented services. In addition, you can refer to the sample files `sc23send.c` and `sc23rec.c` in the `samples/ccitt/` directory for more information about using connection-oriented services on the CCITT network variant. For the China network variant, see the samples of the same names in the s`amples/china` directory.

Processing Overview

Connection-oriented services enable an application to establish and maintain a *connection* or logical communication path with another application for the purpose of exchanging small and large messages. A small message contains no more than 256 bytes of user data, the maximum amount of user data that fits in a single message. A large message contains between 257 and 8192 bytes of user data, which is more data than can fit in a single message. Therefore, the user data in a large message must be divided into multiple segments, each of which is then transported over the SS7 network in a single message. At the destination, the user data in each of these messages is reassembled to form the original user data, which consisted of a single block of data.

To use connection-oriented services, an application running on the SINAP/SS7 system (the *local application*) must assign certain values to specific fields in its `CA_REG` global variable. Table 3-27 lists and describes the fields in the `CA_REG` global variable.

Table 3-27. `CA_REG` **Global Variable Fields**

Field	Description
`max_user_data_size`	The maximum number of user data block in bytes
`max_connections`	The maximum number of connections for this process

NOTE

The `connections_are_owned` field is internal to the SINAP/SS7 system and should not be modified. See the description of the `register.h` file in Chapter 6 for more information.

Implementing connection-oriented services consists of two distinct types of functions:

- The SCCP-SCCP connection-oriented control (SCOC) process performs management functions, such as establishing and releasing a connection. Management requests and responses are passed between the local application and the SCCP-SCOC by means of interprocess communications (IPC) messages. These requests and responses are passed between SCCP-SCOC and the remote application via messages that SCCP-SCOC sends and receives on behalf of the local application.
- The local and remote applications perform data-communication functions that enable the applications to exchange data directly by passing messages to each other. The SCCP-SCOC process is not involved in this data exchange.

The SCCP-SCOC Process

The SCCP-SCOC process performs the management functions necessary to provide connection-oriented services (for example, assigning a connection ID to each connection and

maintaining information about all active connections). The SCCP-SCOC process is the connection-oriented equivalent to the SCCP management process (SCMG), which performs all of the processing necessary to provide an application with connectionless services.

The SCCP-SCOC process relays connection-management requests and responses between local and remote applications during the connection-establishment and connection-release stages. When a local application wants to establish a connection with a remote application, it does not send a connection request directly to the remote application. Instead, the local application sends the SCCP-SCOC process an IPC message containing the connection request. SCCP-SCOC then writes the connection request to a message, which it forwards to the remote application. Likewise, the SCCP-SCOC process receives the remote application's response as a message, which it translates to an IPC message and sends to the local application.

The Stages of Connection-Oriented Communication

The process of communicating via connection-oriented services consists of the following stages:

- The *connection-establishment stage* occurs when two applications obtain a unique connection ID and establish a connection. During this stage, the local and remote applications do not communicate directly. Instead, the applications communicate via IPC messages.
- The *data-transfer stage* occurs when the applications communicate with each other directly via the connection they established earlier. During this stage, the local application can call CASL functions (`ca_put_sc()` and `ca_get_sc()`) to send messages to and retrieve messages from the remote application.
- The *connection-release stage* occurs when the connection and its associated resources are released.

Maintaining Information on Active Connections

The SINAP/SS7 system stores information on active connections in a segment of shared memory called *local reference memory* (LRM). The LRM is an array of `sccp_lrm_t` structures, each containing information about a single active connection between a local and remote application. The LRM entry contains information such as the SSN of the local and remote applications, the connection ID, and the state of the connection.

Each LRM entry is assigned a *local reference number* (LRN), defined as an `sccp_lrn_t` structure, that serves as an index into the LRM array. When a connection is established, the connection ID is assigned to an LRN to tie the connection to a particular entry in the LRM array. The local and remote applications each have a separate LRM entry. The source LRN identifies the local application's LRM entry, and the destination LRN identifies the remote application's LRM entry.

NOTE

The `sccp_lrm_t` and `sccp_lrn_t` structures are internal to the SINAP/SS7 system. They are defined in the `sccp.h` include file.

Inactive Connections and Releasing LRNs

Applications using a connection might not immediately be aware that the connection was released. Therefore, when a connection is released, the connection's LRN remains unavailable (or *frozen*) for a certain amount of time. During this time period, which is defined by the STGUARD timer, the LRN cannot be assigned to another connection. Freezing an LRN ensures that an LRN being used for one connection is not reassigned to another connection before either application realizes the original connection has been released. For example, suppose that two applications are communicating over a connection identified by a particular LRN. Now suppose that the connection is released and, before either application realizes the connection has gone away, the connection's LRN is reassigned to another connection. In this case, either application might attempt to send a response over a connection that no longer exists.

After the amount of time specified by STGUARD, the LRN becomes available and can be assigned to another connection. Note that a connection's LRN is frozen even when the connection is subsequently released by the local application or refused by the remote application.

A connection's LRN is not released when the connection is released. Instead, the LRN is frozen for the amount of time defined by the STGUARD timer. During this time period, the LRN cannot be assigned to another connection, even if it means that the connection cannot be established because no more LRNs are available.

The `release_lrn` command releases frozen LRNs so they can be assigned to other connections. The command can also display LRN statistics such as the total number of LRNs available, the number of LRNs currently being used, and the number of frozen LRNs. This command is described in Chapter 4 of the *SINAP/SS7 User's Guide* (R8051).

Large Message Segmentation and Reassembly

Connection-oriented services enable applications to exchange small and large messages. A small message contains no more than 256 bytes of user data, the maximum amount of user data that fits in a single message. A large message contains between 257 and 8192 bytes of data, which is more data than can fit in a single message. Therefore, the user data in a large message must be divided into multiple segments, each of which is then transported over the SS7 network as a single message. At the destination, the user data in each single message is reassembled into the original data, which is then considered a single block of data.

The CASL functions, `ca_get_sc()` and `ca_put_sc()`, automatically perform message segmentation and reassembly of large messages. You need only to allocate the memory for the buffer in which to store the user data for the message.

SCCP Connection-Oriented Timers

SCCP connection-oriented timers define the amount of time allowed to perform specific connection-oriented tasks (for example, the timer STCONEST defines the amount of time allowed to establish a connection). Table 3-28 presents information about the SCCP connection-oriented timers. For detailed information about these timers, see ITU-T Recommendation Q.714.

NOTE

In Table 3-28, the column labeled "Q.714 Timer" presents the name of the timer as it is referred to in ITU-T Recommendation Q.714, and the column labeled "SCCP Timer Name" presents the name of the corresponding SINAP/SS7 timer.

Table 3-28. SCCP Connection-Oriented Timers

Q.714 Timer	SCCP Timer Name	Range of Valid Values	Description
T(conn est)	STCONEST	1 - 2 minutes	Connection establishment timer
T(iar)	STIAR	3 - 6 minutes	Receive inactivity timer for inbound messages
T(ias)	STIAS†	1 - 2 minutes	Send inactivity timer for outbound messages
T(reset)	STRESET	10 - 20 seconds	Amount of time to wait for a reset confirm message
T(rel)	STREL	10 - 20 seconds	Amount of time to wait for a release complete message after a disconnect
T(interval)	STRELINT	0 - 60 seconds	Total amount of time to repeat the release message
T(repeat rel)	STRELREP	0 - 20 seconds	Amount of time between each repeated release message
T(guard)	STGUARD	8 - 16 minutes	Amount of time to hold an LRN before assigning it to another connection

† The value of the STIAR timer must be greater than the value of the STIAS timer.

The SINAP/SS7 system tables store the values of the SCCP connection-oriented timers. You access these system tables by means of the MML commands DISPLAY-SYSTAB and

`CHANGE-SYSTAB`. For detailed information about these commands, see the discussions on "Displaying the MTP and SCCP System Tables" and "Changing the System Table Timer and Link-Congestion Threshold Settings" in the *SINAP/SS7 User's Guide* (R8051).

Connection-Oriented Messages and Primitives

This section describes the messages and primitives that the SINAP/SS7 system uses to implement connection-oriented services. It contains the following topics.

- IPC message types
- Connection-oriented control primitives used in IPC messages
- Connection-oriented data primitives used in data MSUs

IPC Message Types

The local application and the SCCP-SCOC process use the following types of IPC messages to convey the information necessary to establish and manage connections.

- *Request messages* (for example, `n_connect_req` and `n_disconnect_req`) are those that the local application sends to the remote application to request a particular action.
- *Indication messages* (for example, `n_connect_ind` and `n_disconnect_ind`) are those that the local application receives, indicating that the remote application is requesting a particular action. For example, the local application receives an `n_connect_ind` message when the remote application wants to establish a connection.
- *Response messages* (for example, `n_connect_res` and `n_reset_res`) are those that the local application sends in response to a remote application's request. For example, the local application sends an `n_connect_res` message to respond favorably to the remote application's request to establish a connection.
- *Confirmation messages* (for example, `n_connect_con` and `n_reset_con`) are those that the local application receives, indicating that the remote application has accepted and acted upon a request. For example, the local application receives an `n_connect_con` message when the remote application agrees to the connection request sent by the local application.

These messages are passed between the local application and the SCCP-SCOC process by means of several CASL structures, which are part of the `sccp_ipc_t` structure (described in Chapter 6). Each IPC message type is defined within a particular `sccp_ipc_t` structure. For example, a connection-request message is defined in the `scoc_con_req_t` structure, and a request for a connection ID is defined in the `scoc_get_connid_t` structure.

Connection-Oriented Control Primitives Used in IPC Messages

Table 3-29 and Table 3-30 briefly describe the connection-oriented control primitives used in the IPC messages passed between the local application and the SCCP-SCOC process. These primitives define the IPC message type. For detailed information about any of these primitives, see ITU-T Recommendation Q.712.

Table 3-29 describes the connection-control primitives that you can include in the IPC messages that the local application sends to SCCP-SCOC, and the `sccp_ipc_t` structure in which the IPC message is defined. When sending one of these IPC messages to SCCP-SCOC, your application must set the IPC message's `sccp_ipc_t.i_block_t.ipc_trans_t.msg_type` field to one of the values listed in the column labeled "Primitive." In addition, your application must initialize the `sccp_ipc_t` structure listed in the corresponding column labeled "Structure."

Table 3-29. Outgoing Connection-Control Primitives

Primitive	Structure	Purpose
`I_N_CONNECT_REQ`	`scoc_con_req_t`	Request to establish a connection with a remote application
`I_N_CONNECT_RES`	`scoc_con_res_t`	Response to accept a remote application's request to establish a connection
`I_N_RESET_REQ`	`scoc_res_req_t`	Request to initiate a reset procedure to reinitialize sequence numbers for the connection (class-3 services only)
`I_N_RESET_RES`	`scoc_res_res_t`	Response to accept a remote application's request to initiate a reset procedure to reinitialize sequence numbers for the connection (class-3 services only)
`I_N_DISCONNECT_REQ`	`scoc_dis_req_t`	Request to disconnect (release) the connection
`I_SCOC_GET_CONNID`	`scoc_get_connid_t`	Request to obtain a connection ID for a connection

Table 3-30 describes the connection-control primitives used in the IPC messages that the local application receives from SCCP-SCOC, along with the `sccp_ipc_t` structure in which the IPC message is defined. For incoming IPC messages, your application should examine the value of the `i_block_t.ipc_trans_t.msg_type` field to determine whether the message is a connection-oriented message. If the field's value matches one of the values in the column labeled "Primitive," the message is a connection-oriented message. The application should then read the message by examining the `sccp_ipc_t` structure listed in the corresponding column labeled "Structure."

Table 3-30. Incoming Connection-Control Primitives

Primitive	Structure	Purpose
I_N_CONNECT_CON	scoc_con_con_t	Remote response accepting a local application's connection request
I_N_CONNECT_IND	scoc_con_ind_t	Remote request to establish a connection
I_N_RESET_CON	scoc_res_con_t	Remote response to accept a local application's request to initiate a reset procedure to reinitialize sequence numbers for the connection (class-3 services only)
I_N_RESET_IND	scoc_res_ind_t	Remote request to initiate a reset procedure to reinitialize sequence numbers for the connection (class-3 services only)
I_N_DISCONNECT_IND	scoc_dis_ind_t	Remote request to disconnect (release) the connection; can be a response to a connection request
I_SCOC_CID_RESULT	scoc_cid_result_t	SCCP-SCOC response to request for connection ID

Connection-Oriented Data Primitives Used in Data MSUs

Table 3-31 and Table 3-32 list the connection-oriented data primitives used in the data MSUs passed between local and remote applications. The primitives define the type of data in the MSU. For detailed information about any of these primitives, see ITU-T Recommendation Q.712.

Table 3-31 describes the connection-oriented data primitives that you can include in the data MSUs that the local application sends to the remote application. Your application must set the field `m_block_t.ud_ccitt_msu_t.mtp_ud.ccitt_sccp_user_t.msg_type` to one of the values in the column labeled "Primitive" to define the MSU's data type.

In addition, your application must initialize the `sccp_ipc_t` structure listed in the corresponding column labeled “Structure.”

Table 3-31. Outgoing Connection-Oriented Data Primitives

Primitive	Structure	Purpose
`SC_DATA_FORM1`	`sccp_dt1_t`	Sends a data-form-1 message
`SC_DATA_FORM2`	`sccp_dt2_t`	Sends a data-form-2 message
`SC_EXPEDITED_DATA`	`sccp_expdata_t`	Sends expedited data, which is a data-form-2 message that bypasses the flow-control settings defined for the connection

Table 3-32 describes the connection-oriented data primitives used in the data MSUs received by the local application. Your application should examine the value of the `m_block_t.ud.ccitt_msu_t.mtp_ud.ccitt_sccp_user_t.msg_type` field. If the field’s value matches one of the values in the column labeled “Primitive,” the MSU contains data for a connection-oriented message. The application should then read the data by examining the `sccp_ipc_t` structure listed in the corresponding column labeled “Structure.”

Table 3-32. Incoming Connection-Oriented Data Primitives

Primitive	Structure	Purpose
`SC_DATA_FORM1`	`sccp_dt1_t`	Contains a data-form-1 message
`SC_DATA_FORM2`	`sccp_dt2_t`	Contains a data-form-2 message
`SC_EXPEDITED_DATA`	`sccp_expdata_t`	Contains expedited data, which is a data-form-2 message that bypasses the flow-control settings defined for the connection
`SC_RESET_REQUEST`	`sccp_resetreq_t`	Remote request to initiate a reset procedure to re-initialize sequence numbers
`SC_RELEASED`	`sccp_rlsd_t`	Remote request to release the connection and associated resources

Defining Connection-Oriented Structures

The following CASL structures are used for connection-oriented services. Chapter 6 describes each structure in detail.

- `sccp_ipc_t` structure—Passes IPC messages between the local application and the SCCP-SCOC process. This structure contains several structures, each of which passes a particular type of message.
- `sccp_prim_t` structure—Conveys information about large messages, such as the message size and buffer location. This is an internal structure.
- `sccp_cldclg_t` structure—Contains information about the SCCP called- or calling-party address for a connection-oriented message.
- `sccp_dt1_t` structure—Transports a data-form-1 message.
- `sccp_dt2_t` structure—Transports a data-form-2 message.
- `sccp_expdata_t` structure—Transports a message containing expedited data.

When sending either an IPC message to the SCCP-SCOC process or a data MSU to another application, your application must initialize the appropriate structures. When your application is processing an incoming MSU or IPC message, the structures will have been initialized by the other application, the SCCP-SCOC process, or the SINAP/SS7 system.

In the descriptions of structure fields in Chapter 6, ‘‘CASL Function Calls,” *input* indicates a field (and possibly a corresponding structure) that your application must initialize, and *output* indicates a field (and possibly a corresponding structure) that will have been initialized by the other application, SCCP-SCOC, or the SINAP/SS7 system.

Activating Connection-Oriented Services

To activate connection-oriented services on a SINAP/SS7 node, you **must** define the environment variables described in Table 3-33 at a UNIX system prompt before starting the SINAP/SS7 system on a node. To define these variables automatically each time you start the SINAP/SS7 system, add the variables to the `$SINAP_HOME/Bin/sinap_env.[csh or sh]` file.

Table 3-33. Environment Variables for LRNs

Environmental Variable	Description
SINAP_TOTAL_LR_MEMS=nnnn	Defines the number of local reference memory (LRM) structures specified by nnnn (up to 2000).
SINAP_USER_LR_MEMS=nnnn	Defines the maximum number of connections (up to 2000), specified by nnnn, that an application can have open at any time. **Note:** Every connection requires an LRM; therefore, regardless of the value specified for SINAP_USER_LR_MEMS, the SINAP/SS7 system will not establish more connections than the number of LRMs defined by SINAP_TOTAL_LR_MEMS.
SINAP_TOTAL_LR_NUMS=nnnn	Allocates the number of LRMs specified by nnnn (up to 5000). To accommodate the amount of time a local reference number (LRN) is frozen after use. Stratus recommends allocating more LRNs than LRMs.
SINAP_LRN_FREEZE_TIMEOUT=nnnn	Specifies the number of seconds (up to 1800) before an unused LRN is released and can be assigned to another LRM structure.

Implementing Connection-Oriented Services in an Application

This section provides the background information necessary to design and develop SCCP applications that use connection-oriented services. It contains two subsections, "Application Design Considerations" describes the issues to consider as you design and develop your application and "Summary of Connection-Oriented Application Processing" summarizes the tasks your application must perform to use connection-oriented services.

Application Design Considerations

You should consider the following issues as you design and develop applications that use connection-oriented services.

- When your application sends or receives expedited data (that is, a data MSU of the type SC_EXPEDITED_DATA), the CASL automatically sends a copy of the expedited data to the SCCP-SCOC process. This is necessary so that SCCP-SCOC can issue an expedited-data acknowledgment (SC_CTRL_EXPD_ACK) on behalf of your application. Note that your application need not perform any action to send a copy of the data to SCCP-SCOC nor to issue the expedited-data acknowledgment.

- If your application will handle large messages (that is, messages that are 257 to 8192 bytes in length), consider the following:
  - The `ca_get_sc()` and `ca_put_sc()` functions automatically perform the necessary message segmentation and reassembly. However, your application must allocate and manage the memory for the buffers used to store the message's user data.
  - The `sccp_prim_t` structure conveys information about large messages. The structure's `*p_user_data` field is a pointer to the memory buffer that contains the message's user data, and the structure's `user_data_size` field indicates the data's size. In addition, the `more_data_ind` field indicates whether the message has more user data than can fit in a single MSU. If the user data fits in a single MSU, the value of this field is `0`; otherwise, the value of this field is `1`, which indicates that subsequent MSUs contain additional user data.
  - Only an application's control process can handle large messages; the application's data processes cannot. To handle large messages, an application process **must** register with the SINAP/SS7 system with the `CA_REG` variable's `ss7_primitive` field set to `SS7_CTRL_DATA_PRIMITIVE` or to the value `3`. (Registering in this manner ensures that all segments of a large message are handled by the same application instance.)

Summary of Connection-Oriented Application Processing

The following list describes the tasks a local application must perform to implement connection-oriented services, each of which is described in the following sections.

1. Register for connection-oriented services.
2. Obtain a connection ID to use for the connection.
3. Request a connection with the remote application.
4. Respond to a connection request from a remote application.
5. Begin sending data if the remote application accepts the connection request.
6. Retrieve data sent by the remote application.
7. Release the connection once the data transaction has ended.

> NOTE
>
> The application also must perform several other tasks, such as sending a user in service (UIS) message to SCCP management and terminating processing.

The following sections describe how to perform each task listed above. The descriptions each contain a programming example that shows the application logic for performing a particular connection-oriented task. These programming examples are for illustrative purposes only, and your application logic might differ from that shown in the example. Note that the examples are taken from the sample programs, `sc23common.c`, `sc23send.c`, and `sc23recv.c`, which

are located in the directory: `$SINAP_MASTER/Include/Samples/ccitt`. The SCCP connection-oriented services include file `sc23.h` is also located in this directory.

Task 1: Registering for Connection-Oriented Services

As part of the process of registering with the SINAP/SS7 system, a local application initializes the `CA_REG` global variable, which has a type definition of `register_req_t` structure. The `CA_REG` variable contains registration parameters that define the application's operating characteristics.

To use connection-oriented services, an application must assign the values listed below to the following `CA_REG` fields. Note that the application also must assign values to the other `CA_REG` fields, which are described in the `ca_register()` function's `register_req_t` structure description in Chapter 6 of this manual.

- For `ss7_input_boundary`, specify the value `SS7_INPUT_BOUNDARY_SCCP23`.
- For `max_user_data_size`, specify the maximum allowable size (in bytes) of user data in large messages.
- For `max_connections`, specify the maximum number of simultaneous connections allowed for this application process, up to a maximum of 1000.

NOTE

Based on the values of `max_user_data_size` and `max_connections`, the SINAP/SS7 system creates a pool of memory buffers in which to store the user data for large messages.

Task 2: Obtaining a Connection ID to Use for the Connection

To begin a communication session with a remote application, the local application must obtain a connection ID from the SCCP-SCOC process by performing the following steps:

A. Obtain the IPC key for the SCCP-SCOC process by calling the CASL function `ca_get_key()`, as shown in the following example. (The values `AN_SC` and `PN_SCOC` define the application and process names, respectively, for the SCCP-SCOC process.)

```
ca_get_key(0, 0, AN_SC, PN_SCOC, 0, &ipc_key_t);
```

B. Define the IPC message by initializing the following structure fields:

- For the `sccp_ipc_t.i_block_t.dest_id` field, specify the IPC key returned by `ca_get_key()` to identify the SCCP-SCOC process as the IPC message's destination.
- For the `sccp_ipc_t.i_block_t.ipc_trans_t.msg_type` field, specify the value `I_SCOC_GET_CONNID`. This particular message tells SCCP-SCOC to return the next available connection ID.

- For the `sccp_ipc_t.scoc_get_connid_t.ssn` field, specify the application's SSN to link the connection ID to the local application.

C. Call the function `ca_put_msg()` to deliver the `I_SCOC_GET_CONNID` message to SCCP-SCOC.

Figure 3-6 shows a sample program module that requests a connection ID. Each figure callout corresponds to one of the preceding steps.

```
request_connid()
{
   static sccp_ipc_t sc_prim;
   ipc_key_t sccp_key;

   memset((char *)&sc_prim, 0, sizeof(sc_prim));
A  if ( ca_get_key( 0, 0, AN_SC, PN_SCOC, 0, &sccp_key ) == -1 )
      printf("scsend23: ca_get_key: %s\n", CA_ERR);

   sc_prim.iblock.dest_id = sccp_key;
B  sc_prim.iblock.trans.msg_type = I_SCOC_GET_CONNID;
   sc_prim.primitives.get_connid.ssn = MySSN;
   sc_prim.iblock.msg.len = sizeof(sc_prim) - sizeof(i_block_t);

C  if ( ca_put_msg( (i_block_t *)&sc_prim, 10 ) == -1 )
      printf("scsend23: ca_put_msg: %s\n", CA_ERR);
   else
      printf("scsend23: sent conn_id request to scoc\n");

        •
        •
        •
```

Figure 3-6. Requesting a Connection ID

The local application waits for SCCP-SCOC to respond with the connection ID by performing the following steps:

A. Call the CASL function `ca_get_msg()` to read the IPC queue, waiting for an IPC message whose `i_block_t.ipc_trans_t.msg_type` field is set to `I_SCOC_CID_RESULT`.

B. On receipt of such an `I_SCOC_CID_RESULT` message, check the `sccp_ipc_t.scoc_cid_result_t.conn_id` field, which contains the connection ID.

Figure 3-7 shows a sample program module that implements the preceding programming logic using a `switch` statement. Each figure callout corresponds to one of the preceding steps.

```
void empty_ipc_queue( U16 source )
{
   int i;
   int ret_status = RET_OK;
   BOOL fwait = FALSE; /* FALSE=do not wait, TRUE=wait forever */
   union ib_s {
      i_block_t ipc_header;
      mtp_pause_resume_t pc_ind;
      mtp_status_t congestion_ind;
      scmg_ipc_t sc_state;
      sccp_ipc_t sc_prim;
   } ib;

while (TRUE)
      {
      /* get any msg type; store msg in ib */
A     ret_status = ca_get_msg(0, (i_block_t *)&ib, sizeof(ib), fwait );

      switch ( ib.ipc_header.trans.msg_type )
            {
             case I_MTP_PAUSE:
               printf ("sc23: MTP-PAUSE primitive received, dpc:%lu\n",
                        ib.pc_ind.dpc );
               break;

             case I_SCOC_CID_RESULT:
B              ConnId = ib.sc_prim.primitives.cid_result.conn_id;

   /* if conn_id is negative, the get connection id failed      */
   /* most likely the system has no more lrn entries            */
   /* set ValidId to YES to indicate that a result was received */

               ValidId = YES;
               break;

                  •
                  •
                  •

            }
      }
}
```

Figure 3-7. Obtaining the Connection ID

Task 3: Requesting a Connection with the Remote Application

A local application requests a connection with the remote application by performing the following steps.

A. Obtain the IPC key for the SCCP-SCOC process by calling the CASL `ca_get_key()` function.

B. Define the IPC message by initializing the following structure fields.

- For the `sccp_ipc_t.i_block_t.dest_id` field, specify the IPC key of the SCCP-SCOC process.
- For the `sccp_ipc_t.i_block_t.ipc_trans_t.msg_type` field, specify the value `I_N_CONNECT_REQ`. This particular message tells SCCP-SCOC that you want to establish a connection with the remote application whose addressing information you will provide in Step C.
- For the `sccp_ipc_t.primitives.scoc_con_req_t.conn_id` field, specify the connection ID obtained earlier in "Obtaining a Connection ID." If the remote application accepts the connection request, SCCP-SCOC assigns this connection ID to the connection.

C. Define addressing information for the remote application by assigning values to the fields in the `sccp_ipc_t.primitives.scoc_con_req_t.sccp_cldclg_t` structure.

D. Define the local application's own addressing information by assigning values to the fields in the `sccp_ipc_t.primitives.scoc_con_req_t.sccp_cldclg_t` structure.

E. Define the type of connection you want to establish by assigning appropriate values to the fields in the `sccp_ipc_t.primitives.scoc_con_req_t` structure.

F. Call the function `ca_put_msg()` to deliver the `I_N_CONNECT_REQ` message to the SCCP-SCOC process, which will then forward an MSU to the remote application.

Figure 3-8 shows a sample program module that requests a connection with the remote application. Each figure callout corresponds to one of the preceding steps.

```
void send_n_connect_req( U16 conn_id )
{
   ipc_key_t sccp_key;
   sccp_ipc_t sc_prim;

   /* send connect request ipc msg to scoc process */
   memset((char *)&sc_prim, 0, sizeof(sc_prim));
A  if ( ca_get_key( 0, 0, AN_SC, PN_SCOC, 0, &sccp_key ) == -1 )
      printf("scsend23 ca_get_key: %s\n", CA_ERR);

   sc_prim.iblock.dest_id = sccp_key;
B  sc_prim.iblock.trans.msg_type = I_N_CONNECT_REQ;
   sc_prim.primitives.n_connect_req.conn_id = conn_id;

   /* setup called party address */
   sc_prim.primitives.n_connect_req.called_address.pc_ind = 1;
   sc_prim.primitives.n_connect_req.called_address.pc = RemotePC;
   sc_prim.primitives.n_connect_req.called_address.ssn_ind = 1;
C  sc_prim.primitives.n_connect_req.called_address.ssn = RemoteSSN;
   sc_prim.primitives.n_connect_req.called_address.gti_len = 0;
   sc_prim.primitives.n_connect_req.called_address.rtg_ind = 1;
   sc_prim.primitives.n_connect_req.called_address.national = 0;

   /* setup calling party address */
   sc_prim.primitives.n_connect_req.calling_address.pc_ind = 1;
   sc_prim.primitives.n_connect_req.calling_address.pc =
      PSTATIC->static_dt.own_SPC;
   sc_prim.primitives.n_connect_req.calling_address.ssn_ind = 1;
D  sc_prim.primitives.n_connect_req.calling_address.ssn = MySSN;
   sc_prim.primitives.n_connect_req.calling_address.gti_len = 0;
   sc_prim.primitives.n_connect_req.calling_address.rtg_ind = 1;
   sc_prim.primitives.n_connect_req.calling_address.national = 0;

   sc_prim.primitives.n_connect_req.data_ack_used = 1;
   sc_prim.primitives.n_connect_req.exp_data_used = 1;
   sc_prim.primitives.n_connect_req.qos_class = ClassOfServ;
   sc_prim.primitives.n_connect_req.qos_flow = 4;
E  sc_prim.primitives.n_connect_req.ud_len = 0;
      printf("Class Of Service in N_CONNECT_REQ: %d\n",
      sc_prim.primitives.n_connect_req.qos_class);
   sc_prim.iblock.msg.len = sizeof(sc_prim) - sizeof(i_block_t);

   if ( ca_put_msg( (i_block_t *)&sc_prim, 10 ) == -1 )
F     printf("scsend23: ca_put_msg: %s\n", CA_ERR);
   else
      printf("scsend23: sent n_connect_req to scoc, conn_id:%d\n",conn_id);
}
                  •
                  •
                  •
```

Figure 3-8. Sending a Connection Request

The local application must perform the following steps to determine whether the remote application has accepted the connection request.

A. Call the CASL function `ca_get_msg()` to read the IPC queue and wait for an IPC message whose `i_block_t.ipc_trans_t.msg_type` field is set to `I_N_CONNECT_CON` or `I_N_DISCONNECT_IND`. These particular messages indicate whether the remote application accepted or rejected the connection request.

B. This step is optional. Provide error-handling logic to handle instances when `ca_get_msg()` cannot retrieve an IPC message.

C. Receipt of an `I_N_CONNECT_CON` message indicates that the remote application accepted the connection request. The local application can begin sending data to the remote application, as described later in this chapter.

D. Receipt of an `I_N_DISCONNECT_IND` message indicates that the remote application did not accept the connection request. The local application can perform steps to determine why the remote application refused the connection request.

Figure 3-9 shows how to implement the preceding programming logic using a `switch` statement. Each figure callout corresponds to one of the preceding steps.

```
void empty_ipc_queue( U16 source )
{
   int i;
   int ret_status = RET_OK;
   BOOL fwait = FALSE; /* FALSE=do not wait, TRUE=wait forever */
   union ib_s {
      i_block_t ipc_header;
      mtp_pause_resume_t pc_ind;
      mtp_status_t congestion_ind;
      scmg_ipc_t sc_state;
      sccp_ipc_t sc_prim;
   } ib;

while (TRUE)
   {
   /* get any msg type; store msg in ib */
   ret_status = ca_get_msg(0, (i_block_t *)&ib, sizeof(ib), fwait );
A
   if ( ret_status == -1 )
B     {
       if (errno != ENOMSG)
            printf("mtprecv: %s\n", CA_ERR);
       break;
      }
   else
      {
       switch ( ib.ipc_header.trans.msg_type )
            {
             case I_N_CONNECT_CON:
               DT2SeqNo = 0;
C              SendId = 0;
               RecvId = 0;
               ValidConn = YES;
               break;

             case I_N_DISCONNECT_IND:
               ConnId = 0;
D              DisConn = YES;
               ValidId = NO;
               ValidConn = NO;
               break;

                  •
                  •
                  •

            }
      }
}
```

Figure 3-9. Retrieving a Response to a Connection Request

Task 4: Responding to a Connection Request From a Remote Application

The following steps describe the tasks that a local application must perform to retrieve and respond to a connection request from another application.

A. Call the CASL function `ca_get_msg()` to read the IPC queue, waiting for an IPC message whose `i_block_t.ipc_trans_t.msg_type` field is set to `I_N_CONNECT_IND`. This message type is a connection request from the remote application.

B. This step is optional. Provide error-handling logic to handle instances when `ca_get_msg()` cannot retrieve an IPC message.

C. Receipt of an `I_N_CONNECT_IND` message indicates that the remote application wants to establish a connection with the local application.

D. Save the connection ID to use in subsequent responses.

E. This step is optional. Check to see whether to negotiate the class-of-service (COS) and flow-control parameter values with the remote application. (Note that Figure 3-10 does not contain the programming logic for performing parameter negotiation.)

F. Respond to the remote application's connection request.

- Send an `I_N_CONNECT_RES` message to the SCCP-SCOC process to indicate acceptance of the remote application's connection request. One method of doing this is to call the program module `send_n_connect_res`, as shown in Figure 3-9. (The section, "The send_n_connect_res Program Module," later in this chapter describes this program module.)
- Send an `I_N_DISCONNECT_REQ` message to the SCCP-SCOC process to indicate rejection of the remote application's connection request.

Figure 3-10 shows how to implement the preceding programming logic using a `switch` statement. Each figure callout corresponds to one of the preceding steps.

```
void empty_ipc_queue( U16 source )
{
   int i;
   int ret_status = RET_OK;
   BOOL fwait = FALSE;       /* FALSE=do not wait, TRUE=wait forever */
   union ib_s {
      i_block_t ipc_header;
      mtp_pause_resume_t pc_ind;
      mtp_status_t congestion_ind;
      scmg_ipc_t sc_state;
      sccp_ipc_t sc_prim;
   } ib;

   while (TRUE)
      {
      /* get any msg type; store msg in ib */
      ret_status = ca_get_msg(0, (i_block_t *)&ib, sizeof(ib), fwait );
      if ( ret_status == -1 )
A        {
         if (errno != ENOMSG)
            printf("mtprecv: %s\n", CA_ERR);
B       break;
         }
      else
         {
         switch ( ib.ipc_header.trans.msg_type )
            {
            case I_N_CONNECT_IND:
               if (Mode == SINGLESTEP)
                   printf("sc23: N_CONNECT_IND received, conn_id:%u\n",
C                              ib.sc_prim.primitives.n_connect_ind.conn_id );
               ConnId =  ib.sc_prim.primitives.n_connect_ind.conn_id;
               if (Negotiation == NO)
               {
D                 ClassOfServ = ib.sc_prim.primitives.n_connect_ind.qos_class;
                  FlowControl = ib.sc_prim.primitives.n_connect_ind.qos_flow;
               }
              send_n_connect_res(ib.sc_prim.primitives.n_connect_ind.conn_id);
E               DT2SeqNo = 0;
               SendId = 0;
               RecvId = 0;
               ValidConn = YES;
               break;
                 •
F                •
                 •
            default:
               printf("sc23: unexpected IPC msg, msgtype=%ld\n",
                       ib.ipc_header.trans.msg_type);
            }
         }
}
```

Figure 3-10. Responding to a Connection Request

The `send_n_connect_res` Program Module

The following steps describe the tasks that a local application must perform to accept a connection request from a remote application.

A. Obtain the IPC key for the SCCP-SCOC process by calling the CASL `ca_get_key()` function.

B. Define the IPC message by initializing the following structure fields.

- For the `sccp_ipc_t.i_block_t.dest_id` field, specify the IPC key of the SCCP-SCOC process.
- For the `sccp_ipc_t.i_block_t.ipc_trans_t.msg_type` field, specify the value `I_N_CONNECT_RES`. This message tells SCCP-SCOC that you want to accept the remote application's connection request.
- For the `sccp_ipc_t.primitives.scoc_con_res_t.conn_id` field, specify the connection ID retrieved earlier in Task 4: Responding to a Connection Request from a Remote Application.
- As an optional step, you can negotiate the connection's COS and flow-control parameter values with the remote application by assigning values to other fields in the `sccp_ipc_t.primitives.scoc_con_res_t` structure.

C. Call the function `ca_put_msg()` to deliver the `I_N_CONNECT_RES` message to the SCCP-SCOC process, which will then forward an MSU to the remote application.

Figure 3-11 shows the `send_n_connect_res` program module, which accepts a remote application's connection request. Each figure callout corresponds to one of the preceding steps.

```
  void send_n_connect_res( U16 conn_id )
  {
     ipc_key_t sccp_key;
     sccp_ipc_t sc_prim;

     /* send connect response (confirm) IPC msg to SCOC process */
     memset((char *)&sc_prim, 0, sizeof(sc_prim));
A    if ( ca_get_key( 0, 0, AN_SC, PN_SCOC, 0, &sccp_key ) == -1 )
        printf("sc23send: ca_get_key: %s\n", CA_ERR);

     sc_prim.iblock.dest_id = sccp_key;
     sc_prim.iblock.trans.msg_type = I_N_CONNECT_RES;
     sc_prim.primitives.n_connect_res.conn_id = ConnId;
     sc_prim.primitives.n_connect_res.resp_address = 0;
B    sc_prim.primitives.n_connect_res.data_ack_used = NO;
     sc_prim.primitives.n_connect_res.exp_data_used = YES;
     sc_prim.primitives.n_connect_res.qos_class = ClassOfServ;
     sc_prim.primitives.n_connect_res.qos_flow = FlowControl;
     sc_prim.primitives.n_connect_res.ud_len = 0;
     sc_prim.iblock.msg.len = sizeof(sc_prim) - sizeof(i_block_t);

     printf("Class Of Service: %d, in N_CONNECT_RES\n"
            ,sc_prim.primitives.n_connect_res.qos_class);

C    if ( ca_put_msg( (i_block_t *)&sc_prim, 10 ) == -1 )
        printf("sc23send: ca_put_msg: %s\n", CA_ERR);
     else
        printf("sc23send: sent n_connect_res to scoc\n");
  }
                    .
                    .
                    .
```

Figure 3-11. The `send_n_connect_res` Program Module

Task 5: Begin Sending Data if the Remote Application Accepts the Connection Request

Once a connection is established, the local application can begin sending data MSUs to the remote application by performing the following steps.

A. Define the type of MSU to send by initializing the `m_block_t.sccp_ctrl_t` structure fields, as follows:

- For the `sccp_ctrl` field, specify the value `SC_CTRL_DATA_REQ`.

- For the `sccp_source` field, specify the value `SC_USER` to indicate that the local application is an SCCP user.

B. Call the `CCITT_CA_SET_LABEL` macro to create the MTP routing label for the MSU.

C. Define the destination for the data MSU by specifying the connection ID as the value of the `m_block_t.sccp_prim_t.conn_id` field.

D. For `m_block_t.ud.ccitt_msu_t.mtp_ud.ccitt_sccp_user_t.msg_type`, specify one of the following values to define the type of data MSU you want to send: `SC_DATA_FORM1`, `SC_DATA_FORM2`, or `SC_EXPEDITED_DATA`.

E. Initialize one of the following SCCP data structures to define the MSU data. (Note that Figure 3-12 shows how to send a data-form-1 MSU.)

- For data-form-1 MSUs (SCCP protocol class 2), use the `sccp_dt1_t` structure and specify the value `SC_DATA_FORM1` for the structure's `msg_type` field. Specify the MSU data in the `ud` field, and specify its length in the `ud_len` field.
- For data-form-2 MSUs (SCCP protocol class 3), use the `sccp_dt2_t` structure and specify the value `SC_DATA_FORM2` for the structure's `msg_type` field. Specify the MSU data in the `ud` field, and specify its length in the `ud_len` field.
- For expedited-data MSUs (which are data-form-2 MSUs that enable you to disable flow-control settings), use the `sccp_expdata_t` structure and specify the value `SC_EXPEDITED_DATA` for the structure's `msg_type` field. Specify the MSU data in the `ud` field, and specify its length in the `ud_len` field.

NOTE

To assemble the MSU, the CASL takes the information in the SCCP data structure and writes it to the appropriate fields in the `m_block_t` structure. For example, the user data you define in the `sccp_dt1_t` structure's `ud` field is written to the `ud` field of the `sccp_user_t` structure (which is part of the `m_block_t` structure).

F. For the `m_block_t.mtp_ctrl_t.msg_size` field, specify the length of the MSU's user data, which is equal to the length of the `ud` field of the SCCP data structure plus eight bytes. (For example, if the `ud` field is 255 bytes, define the length of the MSU as 263 bytes.)

G. Call the CASL `ca_put_sc()` function, passing the `m_block_t` structure initialized in the preceding steps.

Figure 3-12 shows a sample program module that sends a data MSU to the remote application. Each figure callout corresponds to one of the preceding steps.

```
void send_dt1( U16 conn_id )
{
   int ret_status, i, j, m;
   m_block_t n, *p_m;
   sccp_dt1_t *p_dt1;

   p_m = &n;
   memset((char *)p_m, 0,sizeof(m_block_t));

   /* send outgoing SC_DATA_FORM MSU */
   p_m->sc_ctrl.sccp_ctrl = SC_CTRL_DATA_REQ;
A  p_m->sc_ctrl.sccp_source = SC_USER;

B  CCITT_CA_SET_LABEL( p_m->ud.ccitt_msu.label, RemotePC,
                              PSTATIC->static_dt.own_SPC, 0 );

C  p_m->sc_prim.conn_id = conn_id;

D  p_dt1 = (sccp_dt1_t*)&(p_m->ud.msu.mtp_ud.sccp.msg_type);

   p_dt1->msg_type = SC_DATA_FORM1;
   p_dt1->ud[0] = 1;    /* BID */
E  p_dt1->ud[1] = DT2SeqNo;
   p_dt1->ud[2] = SendId;
   p_dt1->ud[3] = RecvId;

   j = 4;

   /* fill the data record with printable ASCII characters */
   for (m = 1; m <= 2; m++)
     {
     for (i = 32; i <= 126; i ++)
       p_dt1->ud[j++] = i;
     }
   p_dt1->ud_len = j++;
   DT2SeqNo += 1;

   /* 8 + total_sccp_msg_length for CCITT */
F  p_m->mtp_ctrl.msg_size = 8 + sizeof(sccp_dt1_t);

G  ret_status = ca_put_sc( p_m );
   if ( ret_status == (int)RET_ERR )
      printf("scsend23: cannot send DT1 to sinap\n");
   else
     printf("scsend23: DT1 sent, conn_id:%d seqno:%d\n", conn_id,
DT2SeqNo-1);
}
                  •
```

Figure 3-12. Sending a Data MSU

Task 6: Retrieving Data from the Remote Application

To process incoming data MSUs from the remote application, the local application should perform the following steps.

A. Call the CASL `ca_get_sc()` function to retrieve an incoming MSU.

B. Determine the type of incoming data MSU by checking the value of the `m_block_t.ud.ccitt_msu_t.mtp_ud.ccitt_sccp_user_t.msg_type` field.

C. On receipt of an `SC_DATA_FORM1` MSU, the local application can respond by returning a data-form-1 MSU, as shown in callouts C1, C2, and C3.

C1. Assign the current connection ID to the `connection_id` parameter, and increment the `RecvId` count.

C2. Obtain the sender's ID either from the buffer pointed to by the `m_block_t.sccp_prim_t.p_user_data` field (if this is a large message) or from the `sccp_dt1_1` structure's `ud` field (if this is a small message).

C3. Call the `send_dt1` program module to send the response MSU, passing the current connection ID as an argument. (The `send_dt1` program module is shown later in this chapter in "The send_dt1 Program Module.")

D. On receipt of an `SC_DATA_FORM2` MSU, the local application can respond by returning a data-form-2 MSU, as shown in callouts D1, D2, and D3.

D1. Assign the current connection ID to the `connection_id` parameter, and increment the `RecvId` count.

D2. Obtain the sender's ID either from the buffer pointed to by the `m_block_t.sccp_prim_t.p_user_data` field (if this is a large message) or from the `sccp_dt2_t` structure's `ud` field (if this is a small message).

D3. Call the `send_dt2` program module to send the response MSU, passing the current connection ID as an argument. (The `send_dt2` program module is shown later in this chapter in "The send_dt2 Program Module.")

E. Receipt of an `SC_EXPEDITED_DATA` MSU indicates the presence of expedited data in the `sccp_expdata_t` structure's `ud` field.

Figure 3-13 shows a sample program module that retrieves an incoming data MSU. Each figure callout corresponds to one of the preceding steps.

```
  void empty_msu_queue( U16 user )
  {
     int i, j;
     int bid, seqno, size;
     int finished, connection_id;
     m_block_t *p_m;
     sccp_expdata_t *p_expd;
     sccp_dt2_t *p_dt2;
     sccp_dt1_t *p_dt1;
     finished = NO;

     while ( !finished )
        {
A       p_m = ca_get_sc( 0 );

        if ( p_m != (m_block_t*)RET_ERR )
           {

B          switch ( p_m->ud.ccitt_msu.mtp_ud.sccp.msg_type )
             {
C               case SC_DATA_FORM1:
                  p_dt1 = (sccp_dt1_t*)&(p_m->ud.msu.mtp_ud.sccp.msg_type);
                  ValidMsg = YES;

  /* respond with a data form 1 message */
                  if (user == RECEIVER)
                     {
C1                   connection_id = p_m->sc_prim.conn_id;
                     RecvId++;

  /* pick up the send identification from either the MSU or large data
  buffer */
                     if (p_m->sc_prim.p_user_data != (U8 *)NULL
                          && p_m->sc_prim.user_data_size != 0)
C2                       SendId = p_m->sc_prim.p_user_data[2];
                     else
                         SendId = p_dt1->ud[2];

  /* do not send a response message if no request to do so is posted */
C3                   if (DataResponse == YES)
                        send_dt1(connection_id);
                     DT2Count += 1;
                     }
                break;
```

```
D             case SC_DATA_FORM2:
                p_dt2 = (sccp_dt2_t*)&(p_m->ud.msu.mtp_ud.sccp.msg_type);
                ValidMsg = YES;

  /* respond with a data form 2 message */
                if (user == RECEIVER)
                   {
D1                 connection_id = p_m->sc_prim.conn_id;
                   RecvId++;

  /* pick up the send identification from either the MSU or large data
  buffer */
D2                 if (p_m->sc_prim.p_user_data != (U8 *)NULL
                        && p_m->sc_prim.user_data_size != 0)
                       SendId = p_m->sc_prim.p_user_data[2];
                   else
                       SendId = p_dt2->ud[2];

  /* do not send a response message if no request to do so is posted */
D3                  if (DataResponse == YES)
                       send_dt2(connection_id);
                   DT2Count += 1;
                   }
              break;

E
              case SC_EXPEDITED_DATA:
                p_expd =
  (sccp_expdata_t*)&(p_m->ud.msu.mtp_ud.sccp.msg_type);
                if (Mode == SINGLESTEP)
                 printf("Expedited Data msu received
  conn_id:%d,ltid:%c%c%c%c\n",
                     p_m->sc_prim.conn_id, p_expd->ud[4],
                     p_expd->ud[5], p_expd->ud[6], p_expd->ud[7] );
                break;
            }
         }
      else if ( errno != CA_ERR_NO_MSUS && errno != EINTR )
         {
         finished = YES;
         printf("msu read error: %d\n", errno );
         }
      else
         finished = YES;
      }
  }
                   •
```

Figure 3-13. Retrieving an Incoming Data MSU

The `send_dt1` Program Module

Figure 3-14 shows the `send_dt1` program module, which retrieves an incoming data-form-1 message from the remote application. For more information on the `send_dt1` program module, see the section, "The send_n_connect_res Program Module," earlier in this chapter.

```
void send_dt1( U16 conn_id )
{
   int ret_status, i, j, m;
   m_block_t n, *p_m;
   sccp_dt1_t *p_dt1;

   p_m = &n;
   memset((char *)p_m, 0,sizeof(m_block_t));

   /* send outgoing SC_DATA_FORM MSU */
   p_m->sc_ctrl.sccp_ctrl = SC_CTRL_DATA_REQ;
   p_m->sc_ctrl.sccp_source = SC_USER;

   CCITT_CA_SET_LABEL( p_m->ud.ccitt_msu.label, RemotePC,
                               PSTATIC->static_dt.own_SPC, 0 );

   p_m->sc_prim.conn_id = conn_id;

   p_dt1 = (sccp_dt1_t*)&(p_m->ud.msu.mtp_ud.sccp.msg_type);
   p_dt1->msg_type = SC_DATA_FORM1;
   p_dt1->ud[0] = 1;    /* BID */
   p_dt1->ud[1] = DT2SeqNo;
   p_dt1->ud[2] = SendId;
   p_dt1->ud[3] = RecvId;

   j = 4;

/* fill the data record with printable ASCII characters */

   for (m = 1; m <= 2; m++)
     {
     for (i = 32; i <= 126; i ++)
       p_dt1->ud[j++] = i;
     }

   p_dt1->ud_len = j++;
   DT2SeqNo += 1;

   /* 8 + total_sccp_msg_length for CCITT */
   p_m->mtp_ctrl.msg_size = 8 + sizeof(sccp_dt1_t);
   ret_status = ca_put_sc( p_m );
   if ( ret_status == (int)RET_ERR )
      printf("sc23send: cannot send DT1 to sinap\n");
   else
      printf("sc23send: DT1 sent, conn_id:%d  seqno:%d\n", conn_id, DT2SeqNo-1);
}
                  .
                  .
                  .
```

Figure 3-14. Retrieving an Incoming Data-Form-1 Message

The `send_dt2` Program Module

Figure 3-15 shows the `send_dt2` program module, which retrieves an incoming data-form-2 message from the remote application. For more information on the `send_dt2` program module, see the section, “The send_n_connect_res Program Module,” earlier in this chapter.

```
void send_dt2( U16 conn_id )
{
   int ret_status, i, j, m;
   m_block_t n, *p_m;
   sccp_dt2_t *p_dt2;

   p_m = &n;
   memset((char *)p_m, 0,sizeof(m_block_t));

   /* send outgoing SC_DATA_FORM MSU */
   p_m->sc_ctrl.sccp_ctrl = SC_CTRL_DATA_REQ;
   p_m->sc_ctrl.sccp_source = SC_USER;

   CCITT_CA_SET_LABEL( p_m->ud.ccitt_msu.label, RemotePC,
                              PSTATIC->static_dt.own_SPC, 0 );

   p_m->sc_prim.conn_id = conn_id;

   p_dt2 = (sccp_dt2_t*)&(p_m->ud.msu.mtp_ud.sccp.msg_type);
   p_dt2->msg_type = SC_DATA_FORM2;
   p_dt2->ud[0] = 1;    /* BID */
   p_dt2->ud[1] = DT2SeqNo;
   p_dt2->ud[2] = SendId;
   p_dt2->ud[3] = RecvId;

   j = 4;

/* fill the data record with printable ASCII characters */

   for (m = 1; m <= 2; m++)
     {
     for (i = 32; i <= 126; i ++)
       p_dt2->ud[j++] = i;
     }

   p_dt2->ud_len = j++;
   DT2SeqNo += 1;

   /* 8 + total_sccp_msg_length for CCITT */
   p_m->mtp_ctrl.msg_size = 8 + sizeof(sccp_dt2_t);
   ret_status = ca_put_sc( p_m );
   if ( ret_status == (int)RET_ERR )
      printf("sc23send: cannot send DT2 to sinap\n");
   else
      printf("sc23send: DT2 sent, conn_id:%d  seqno:%d\n", conn_id, DT2SeqNo-1);
}
                  •
                  •
                  •
```

Figure 3-15. Retrieving an Incoming Data-Form-2 Message

Task 7: Releasing the Connection

After completing the data transaction, the local application should release the connection by performing the following steps. (Note that the remote application can also release the connection.) You may want to include these steps as part of the application's termination process, which includes sending a UOS (user-out-of-service) message to SCCP management and then calling the CASL function `ca_terminate()` to terminate the application's processing.

To release the connection and end the communication session, the local application must perform the following steps.

A. Obtain the IPC key for the SCCP-SCOC process by calling the CASL `ca_get_key()` function.

B. For the `sccp_ipc_t.i_block_t.dest_id` field, specify the IPC key of the SCCP-SCOC process.

C. For the `sccp_ipc_t.i_block_t.ipc_trans_t.msg_type` field, specify the value `I_N_DISCONNECT_REQ`. This particular message tells SCCP-SCOC that you want to release the specified connection with the remote application.

D. Assign appropriate values to the fields in the `sccp_ipc_t.primitives.scoc_dis_req_t` structure to provide the information for the disconnect request. For a normal end, specify the value `SCCP_CAUSE_REL_EUO` for the structure's `reason` field.

E. Call the function `ca_put_msg()` to deliver the `I_N_DISCONNECT_REQ` message to the SCCP-SCOC process.

The disconnect request releases the connection between the local and remote applications, and returns the connection ID to the pool of available connection IDs. Note that the local application need not wait for a response from SCCP-SCOC.

Figure 3-16 shows the `terminate` program module, which terminates application processing. The `terminate` program module calls the `send_n_disconnect_req` program module (also shown), which issues a disconnect request to release a connection.

```
void terminate( U16 source )
{
   static terminate_t term;

   send_nstate_uos_to_sccp();
   ca_withdraw(); /* inform driver not to send any MSUs */
   printf("Terminating this process - wait 15sec\n");
   sleep(15);

/* if this is the send side, then disconnect */
   if (source == SENDER)
     send_n_disconnect_req( ConnId );

   term.ipc_key = CA_KEY;
   term.msg_type = TERM_SELF_INITIATED;
   term.fss = 0;
   strcpy((char *)&term.reason[0], "Client application exiting");
   term.exit_code = 0;

   ca_terminate(&term);
   exit(0);
}
     •
     •
     •

void send_n_disconnect_req( U16 conn_id )
{
   ipc_key_t sccp_key;
   sccp_ipc_t sc_prim;

A  /* send released IPC msg to SCOC process */
   memset((char *)&sc_prim, 0, sizeof(sc_prim));
   if ( ca_get_key( 0, 0, AN_SC, PN_SCOC, 0, &sccp_key ) == -1 )
      printf("sc23send: ca_get_key: %s\n", CA_ERR);

   sc_prim.iblock.dest_id = sccp_key;

   sc_prim.iblock.trans.msg_type = I_N_DISCONNECT_REQ;
   sc_prim.primitives.n_disconnect_req.conn_id = conn_id;
C  sc_prim.primitives.n_disconnect_req.reason = SCCP_CAUSE_REL_EUO;
   sc_prim.primitives.n_disconnect_req.resp_address = 0;
D  sc_prim.primitives.n_disconnect_req.ud_len = 0;

   sc_prim.iblock.msg.len = sizeof(sc_prim) - sizeof(i_block_t);

   if ( ca_put_msg( (i_block_t *)&sc_prim, 10 ) == -1 )
      printf("sc23send: ca_put_msg: %s\n", CA_ERR);
   else
```

Figure 3-16. Releasing the Connection

Load Control

The load control facility, hereafter referred to as *load control*, improves application throughput when there is severe network congestion and reduces the risk that incoming MSUs will be lost or discarded due to timeouts. It can be applied only to applications registered at the TCAP input boundary. It cannot be set up for MTP or SCCP applications. This section provides an overview of load control and describes the considerations of which you should be aware as you design and build TCAP applications in which load control will be implemented. See the *SINAP/SS7 User's Guide* (R8051) for more information about implementing load control.

Load control can be invoked by operator command or programmatically (forced load control) but normally goes into effect due to monitored conditions being met (automatic load control). Load control is optional. It is set up on an application basis and each application can have its own load control parameters. Once set up, load control can be enabled or disabled on a system-wide, application, or instance basis. Load control can take place on a group basis for an application as a whole, or on an individual basis for each separate instance. The setup and enablement parameters for a subsystem are persistent. They are retained in the static database and only need to be reissued when a change is called for.

During normal SS7 network operation, an application processes MSUs in the order in which they arrive. However, when there is extreme network congestion, the application may not be able to handle all of its incoming MSUs; therefore, it is important for the application to complete existing transactions before initiating new ones. To accomplish this, load control allows the SINAP/SS7 system to assign precedence to MSUs that are part of an existing transaction (*continuation MSUs*).

You configure an application for load control by defining the application's load control operating characteristics. These operating characteristics specify how the SINAP/SS7 system is to perform load control processing, and they define such things as the maximum level of network congestion considered acceptable for the application.

After configuring the application, you enable load control, which causes the SINAP/SS7 system to begin monitoring the application's congestion level. There are two options to specify when load control should begin.

The first option involves evaluating the MSU delay count and the input queue length. Each incoming message is given a timestamp that indicates the time of its arrival. The SINAP/SS7 system uses this timestamp to evaluate the length of time between the message arrival and when the application places a response on the output queue. This length of time, as well as the input queue length, is optionally considered in calculating the congestion level. When the congestion level exceeds the threshold level defined by the application's load control operating characteristics (hereafter called *overload conditions*), the SINAP/SS7 system begins load control processing.

The second option for determining load control onset involves disabling the use of MSU delay counts and considering only the input queue length versus the threshold. Incoming MSUs are not timestamped. See Chapter 3 of the *SINAP/SS7 User's Guide* (R8051) for more information.

NOTE

If you are not using MSU delay counts, which use timestamps, the restriction requiring using the same `T_block` for output does not apply.

Performing Load Control Processing

Each SINAP/SS7 application has an input queue on which the SINAP/SS7 system places incoming MSUs. This input queue functions as a *first-in, first-out* (FIFO) queue; hereafter, this queue is referred to as the application's input queue. To perform load control processing, the SINAP/SS7 system creates a second input queue for the application. This queue functions as a *last-in, first-out* (LIFO) queue; hereafter, this queue is referred to as the application's LIFO queue.

At load control onset, the SINAP/SS7 system evaluates the MSUs that are currently on the application's input queue and discards the ones that initiate a new transaction (`BEGIN` and `UNI` for CCITT/TTC/NTT/China; `QUERY` and `UNI` for ANSI). Thereafter, the SINAP/SS7 system places all incoming MSUs on the application's LIFO queue. The SINAP/SS7 system continues to place continuation MSUs (those that are part of an existing transaction) on the application's input queue. MSUs that are not of this category, such as dialogue responses, continue to be placed on the FIFO queue. To make use of the timestamp applied to incoming MSUs, an application **must** use the same `t_block_t` structure for the incoming MSU's response as was used by the incoming MSU; otherwise, the timestamp is rendered useless.

When timestamping is used, each incoming MSU is timestamped when it arrives at that process's input queue. The CASL TCAP functions are modified to save this timestamp, if present, in an array of timestamps paralleling the `tblock` array, as the incoming `mblock` is not preserved. On TCAP output, the saved timestamp, if present, is inserted in the first timeslot in the outgoing `mblock`. When the `mblock` is extracted from the process's output queue in the driver before going to the Link Multiplexor, the difference between the current time and this timestamp is compared with the setup delay limit. If this limit is reached or exceeded, a `prs` counter is incremented and compared against the setup count limit, else the counter is reset. If the timestamp is not present, the counter is also reset.

The application processes the MSUs on its input queue first. When all these MSUs have been processed, the application begins processing the MSUs on its LIFO queue (those that initiate new transactions). If the LIFO queue becomes full, the SINAP/SS7 system discards the oldest MSU on the queue to make room for a new incoming MSU. Messages are always discarded from the LIFO queue when they exceed a given timeout period that has been configured. If the next MSU extracted from the LIFO queue has experienced excessive delay (a setup parameter) and, by implication, so has the rest of the LIFO queue, the queue's contents are discarded.

TCAP MSU `dialogue begins` are considered those that are NOT flagged in the `mblock` as `DR_TO_PID`. This is so that if forced load control is invoked or automatic load control comes into effect, scanning of the FIFO queue for `dialogue begins` can be done rapidly.

The LIFO queue used during load control has a maximum length equal to that specified for the LIFO queue at registration. If an MSU about to be queued on the LIFO queue would cause it to overflow, the oldest MSU on the queue is discarded instead.

An additional restriction is that group load control can be used with either least used or round robin inbound load distribution, whereas individual load control can only be used with round robin, where instance loads could vary widely. Doing an MML or CASL setup with `type=individual`, when the application has registered as least used will result in an error being returned. When the setup parameters are pre-existing (already set up in the static database) and an application then registers, the `ca_register()` function can return with the following new error codes:

- `CA_ERR_REG_NOT_TCAP` - Indicates the input boundary is specified as SCCP, but load control has been set up.
- `CA_ERR_REG_DIST_WRONG` - Indicates the load distribution type is `LEAST_UTILIZED`, but the load control setup type is specified as `individual`.
- `CA_ERR_REG_THRESHOLD` - Indicates the maximum input MSU count is less than the load control threshold value.

The SINAP/SS7 system continues to perform load control processing until the application has processed all MSUs on its input and LIFO queues (for all instances if on a group basis), or until you terminate load control (see the section, “Disabling Load Control for an Application” later in this chapter). When both queues are empty, the SINAP/SS7 system disables load control processing for the application. Since the application is still configured and enabled for load control, the SINAP/SS7 system continues to monitor the application for overload conditions. When the application again experiences overload conditions, the SINAP/SS7 system invokes load control processing for the application.

Load control is persistent and once it is set up for an SSN or application it stays set up even when the application is terminated or the SINAP/SS7 system is restarted. The only way to remove the load control setting for such an application is to delete it by using an MML command (see the sections describing load control in the *SINAP/SS7 User’s Guide* (R8051)) or a CASL function (see Chapter 6, ‘‘CASL Function Calls”).

Alarms of major levels are always issued for all cases of load control and abatement. Alarms are issued separately for instances if individual load control is in effect.

Implementing Load Control Functionality

Any application that interfaces with the SINAP/SS7 system at the TCAP boundary can implement load control functionality. You can configure and enable an application for load control by issuing MML commands from the SINAP/SS7 system’s Terminal Handler, or by including CASL function calls in the application (typically, in the application’s control process).

You can use a combination of MML commands and CASL functions to implement load control for an application. For example, you can develop an application that calls the CASL function

`ca_setup_locon()` but does not call `ca_enable_locon()`. The `ca_setup_locon()` function call configures the application for load control; however, since `ca_enable_locon()` is not called, the application is not enabled for load control. You must issue the MML command `ENABLE-LOAD-CONTROL` to enable load control. If you then want to change the application's load control characteristics, you can issue the `SETUP-LOAD-CONTROL` command rather than have the application call `ca_setup_locon()`, which would necessitate recompiling the application. (See "Load Control Functions" in Chapter 6 for a description of load control CASL functions, and see Appendix A, ''SINAP/SS7 MML Command Summary," for a summary of load control MML commands. The *SINAP/SS7 User's Guide* (R8051) provides detailed information about the Terminal Handler and load control MML commands.)

NOTES

1. Load control cannot be implemented by an application that has one process performing both control and data processing, and another process with one or more instances also performing data processing at the same time. To implement load control, an application must have a separate control process, or it must have a process with several instances, one of which performs control processing.
2. An application should implement load control only in response to extreme network congestion. The use of load control does not guarantee that incoming MSUs will not be lost.

Configuring Load Control

You configure an application for load control by issuing the MML `SETUP-LOAD-CONTROL` command or by having the application call the `ca_setup_locon()` function. By configuring an application for load control, you define the various thresholds that determine the application's maximum allowable congestion level. Other arguments specify how the SINAP/SS7 system is to perform load control processing for the application.

You cannot configure individual application instances for load control because by default, the `SETUP-LOAD-CONTROL` command and the `ca_setup_locon()` function affect all instances of the application. However, when you enable load control, you can enable load control for a **subset** of the application's instances, thereby selectively implementing load control processing for specific application instances.

Enabling Load Control for an Application

After configuring an application for load control, you initiate load control operation. You do this by issuing the MML `ENABLE-LOAD-CONTROL` command or by having the application call the `ca_enable_locon()` function, which causes the SINAP/SS7 system to begin monitoring the application's congestion level. When the application experiences overload

conditions, the SINAP/SS7 system automatically begins performing load control processing for the application.

For the SINAP/SS7 system to implement load control processing, load control must be enabled at each of the following levels.

- *System* – The SINAP/SS7 system is enabled to perform load control processing.
- *Application* – A particular application is enabled to perform load control processing.
- *Instance* – Individual application instances are enabled to perform load control processing.

By default, the MML `ENABLE-LOAD-CONTROL` command and the `ca_enable_locon()` function automatically enable load control at the system and instance levels. You enable load control at the application level by issuing the MML `ENABLE-LOAD-CONTROL` command or having the application call the `ca_enable_locon()` function, specifying the application for which you want load control enabled.

When load control is enabled (at all levels), monitoring of the input and output queues begins. When a threshold is exceeded for the length of the input FIFO queue and, if optionally specified, a running count of consecutive output MSUs is delayed beyond a time limit, load control begins. These conditions must be met by all instances of a subsystem if group load control was specified at setup. The threshold, delay time, and count limit are all parameters set using the `SETUP-LOAD-CONTROL` command described in the *SINAP/SS7 User's Guide* (R8051).

If you disable load control at the system level, you cannot enable load control for a particular application until you first re-enable load control at the system level.

- You disable load control at the system level by issuing the MML `DISABLE-LOAD-CONTROL` command or having the application call the `ca_disable_locon()` function with `SSN=ALL`.
- You re-enable load control at the system level by issuing the MML `ENABLE-LOAD-CONTROL` command or having the application call the `ca_enable_locon()` function with `SSN=ALL`.

Likewise, if you disable load control for specific application instances, you must also explicitly re-enable load control for those instances; enabling load control at the application level does **not** supersede the load control enablement settings for individual application instances. For example, if you disable load control for instances `1`, `3`, and `5` of an application whose specified subsystem number (SSN) is `254`, you must explicitly re-enable load control for those instances; enabling load control for SSN `254` does not enable load control for instances `1`, `3`, and `5`.

Disabling Load Control for an Application

The SINAP/SS7 system automatically terminates load control processing when the application finishes processing all MSUs on its input queue and on the LIFO queue created at the onset of load control processing. However, load control is persistent, and once set up for an application, it stays even when the application terminates or the SINAP/SS7 system is restarted. To remove load control settings for an application, use one of the following commands:

- To terminate load control completely, issue the MML `DISABLE-LOAD-CONTROL` command or have the application call the `ca_disable_locon()` function. The SINAP/SS7 system does **not** return to monitoring the application for overload conditions. To reactivate load control, you must re-enable load control for the application.
- To reconfigure the application and remove load control functionality, issue the MML `SETUP-LOAD-CONTROL` command or have the application call the `ca_setup_locon()` function with `TYPE=DELETE`. The application continues to execute; however, it is no longer configured for load control.

NOTE

When you use either of these methods to terminate load control, the SINAP/SS7 system extracts MSUs from the application's LIFO queue in FIFO fashion and appends them to the application's input queue. The SINAP/SS7 system discards any MSUs that would cause the input queue to overflow.

The Disable Load Control (`DISABLE-LOAD-CONTROL`) command or the `ca_disable_locon()` function terminates load control completely at the system, application, or instance level. The SINAP node does **not** return to monitoring the application for overload conditions. To reactivate load control, you must re-enable load control for the application. You can access this command through the `DISABLE` option of the load control menu.

You can disable load control at the *system* level, the *application* level, or the *instance* level in the following ways:

- Disabling load control at the *system level* terminates load control operation for all applications configured for load control. In the case of a critical situation this allows an operator or control program to disable load control for all applications by using a single command or function call. To disable load control at the system level, issue the `DISABLE-LOAD-CONTROL` command or `ca_disable_locon()` and specify the value `ALL` for the `SSN` argument.

NOTE

After disabling load control at the system level, you must re-enable load control at the same level before you can enable load control for an individual application. (You re-enable load control at the system level by issuing the `ENABLE-LOAD-CONTROL` command and specifying the value `ALL` for the `SSN` argument.)

- Disabling load control at the *application level* terminates load control operation for all instances of a specific application. To disable load control at the application level, issue the `DISABLE-LOAD-CONTROL` command or `ca_disable_locon()`. Specify the SSN

of the application as the value of the SSN argument and specify the value ALL for the INSTANCE argument, or omit INSTANCE from the command line.

- Disabling load control at the *instance level* terminates load control operation for one or more instances of an application. To disable load control at the instance level, issue the DISABLE-LOAD-CONTROL command or ca_disable_locon(). Specify the SSN of the application as the value of the SSN argument and specify the number of the application instance as the value of the INSTANCE argument. To use this form of the DISABLE-LOAD-CONTROL command or ca_disable_locon(), the application must be configured for individual-type operation. (See the description of the ca_disable_locon() TYPE argument in Chapter 6, ‘‘CASL Function Calls.”)

  NOTES

  1. For an instance, disabling load control at any level disables load control for that instance.
  2. When an application stops running, load control cancels the disablement settings of individual application instances.
  3. After disabling load control for specific application instances, you must re-enable load control for those same instances. Enabling load control at the application level does not override the enablement settings for individual application instances. For example, the following command disables load control for instances 1, 3, and 5 of the application whose SSN is 254.

     ```
     DISABLE-LOAD-CONTROL:SSN=254:INSTANCE=1&3&5
     ```

  4. Issuing the command, ENABLE-LOAD-CONTROL:SSN=254, (which enables load control for all instances of SSN 254) does **not** re-enable load control for instances 1, 3, and 5 of SSN 254. To re-enable load control for those instances, you must issue the following command.

     ```
     ENABLE-LOAD-CONTROL:SSN=254:INSTANCE=1&3&5
     ```

If you issue the DISABLE-LOAD-CONTROL command or ca_disable_locon() while the SINAP/SS7 system is performing load control processing for an application, the SINAP/SS7 system extracts MSUs from the application’s LIFO queue in FIFO fashion and appends them to the application’s input queue. The SINAP/SS7 system discards MSUs that have been on the application’s LIFO queue longer than the time defined by the SETUP-LOAD-CONTROL command or ca_disable_locon() ABATEDELAY argument. (See “Enabling Load Control for an Application” earlier in this chapter for how to configure load control parameters for an application.) The SINAP/SS7 system also discards any MSUs that would cause the application’s input queue to overflow.

`EXIT-LOAD-CONTROL` and `ca_exit_locon()` apply only to forced load control. Using them when forced load control is not in effect results in an error message or code. These cause premature abatement of load control in the same manner as disabling load control, except load control stays enabled. `INVOKE-LOAD-CONTROL` or `ca_invoke_locon()` converts automatic load control, if it is in effect, to forced load control with no other changes except that, during forced load control, there is no monitoring of abatement conditions.

Terminating a process results in the driver attempting to redistribute the MSUs on the FIFO and then LIFO queues to other existing instances. This results in abatement of either form of load control then in effect for that instance, which might effect the group. The state of the application remains enabled.

Setup deletion during either form of load control results in premature abatement. The state of the process remains running, but without load control setup.

The setup and enablement parameters in the shared memory static database are saved to disk when a `BACKUP-NODE` command is executed. However, the MML commands to set up, enable, or disable a subsystem, since they affect the static database, are logged to the `rclog` file for possible later playback with a `RESTORE-NODE` command. The application's use of the corresponding CASL function results in building a corresponding MML string and placing it in `rclog`. The setup strings have a `NOTIFY=Y/N` parameter, even though this cannot be entered by the user through MML. The `RESTORE-NODE` command replaces the static database with the last saved version, with optional playback of the MML strings in the `rclog` file. This updates the setup and enablement settings for every running process set up for load control.

Enabling Loopback Detection (CCITT)

For the CCITT network variant only, when the loopback detection environment variable is defined, the SINAP/SS7 system can detect when a remote link is in a loopback mode. During a signaling link test (SLT), normally run after MTP Level 2 alignment, if the system receives signaling test messages (SLTMs) from the remote link that are identical to the SLTMs it sent to that link for all signaling link test attempts, and no correct signaling link test acknowledgment (SLTA) message is received, the SINAP/SS7 system sets a loopback diagnostic indicator to allow display of the loopback status on the `DISPLAY-LINK` screen.

NOTE

The loopback detection feature is active only during the signaling link test procedure.

To enable the loopback detection feature, define the following environment variable before you start or restart the SINAP/SS7 system:

```
LOOPBACK_DISPLAY
```

It is not necessary to assign a value to the variable. The SINAP/SS7 system only verifies the existence of the variable. You can define the environment variable at the UNIX command level before starting or restarting the SINAP node. To automatically define the variable each time you start the SINAP node, uncomment the variable in the `$SINAP_HOME/Bin/sinap_env[.csh` or `.sh]` file.

If you do not define the environment variable, the SINAP/SS7 system does not perform loopback detection and the loopback status is not displayed on the `DISPLAY-LINK` screen.

Enabling Transfer-Restricted Message Handling

For the CCITT network variant, when the SINAP node receives a transfer-restricted (TFR) message, the node starts the MTP Level 3 T10 timer and does not send a signaling route set restricted (RSR) test message immediately. An RSR is sent to the signaling transfer point referring to the destination declared restricted by the TFR message every T10 period until a transfer-allowed (TFA) message is received.

To use the transfer-restricted (national network) option you must define the following environment variable before you start or restart the SINAP node:

```
MTP_WHITE_BOOK_TFR
```

To automatically define this environment variable each time you start or restart the SINAP node, uncomment the variable in the SINAP/SS7 environment file `$SINAP/HOME/Bin/sinap_env[.csh or.sh]`. You do not need to assign a value to the variable, the SINAP/SS7 system simply verifies the existence of the variable.

When the `MTP_WHITE_BOOK_TFR` option is defined, the Display Routeset (`DISPLAY-RSET`) command, accessed through `sysopr`, displays the following additional status states for the selected route sets and routes:

For Route Set Status:	`R` - DPC restricted
	`P` - DPC prohibited
For Route Status:	`R` - Transfer restricted
	`P` - Transfer prohibited

Figure 3-17 shows a situation where LSET1 route is restricted (status of "`R`") and LSET2 route is prohibited (status of "`P`"). The status of the resulting RSET3 route set (or DPC=3003) is "`Rbc`," where "`R`" indicates that DPC is accessible, but restricted.

```
Display Route Set:
                .....
RSet Name = RSET3,  DPC = 3003,  CPC Count of LSSN = 0
State = ACTIVE,  Status = Rbc,  Load Sharing = YES

RouteName  Priority  Status
LSET1             1  aRlr
LSET2             2  aPlr

--Route Set Status Legend--
a - DPC accessible              A - DPC not accessible
                                R - DPC restricted
                                P - DPC prohibited
b - route set not blocked       B - route set blocked
c - route set not congested     C - route set congested

--Route Status Legend--
a - link set available          A - link set not available
x  - transfer allowed           X - transfer not allowed
                                P - transfer prohibited
l - link set not congested      L - link set congested
r - route not congested         R - route congested
```

Figure 3-17. Sample Output with Restricted Message Handling

For the ANSI network variant, the SINAP/SS7 system can process both the transfer-restricted (national network) and transfer-controlled (U.S. networks), but does not allow the transfer-controlled (international networks).

RSR/RSP in Response to TFR/TFP (ANSI)

For the ANSI network variant default setting, when the SINAP/SS7 system receives a TFR or TFP message, it waits until the MTP Level 3 T10 timer expires before sending a signaling route set test for restricted or prohibited destination (RSR or RSP) messages. This behavior conforms to1992/1996 ANSI and ITU-T specifications. However, there is a deviation in the 1988 ANSI Standard's MTP Level 3 SDL diagram, Figure 46/T1.111.4 (RSRT sheet 1 of 3), where an RSR or RSP is responded to right after the TFR or TFP is received by the receiving node, and then after every T10 period. If this behavior is desired, set the environment variable:

```
MTP_ANSI88_RSR_RST
```

To automatically define this environment variable each time you start or restart the SINAP node, uncomment the variable in the SINAP/SS7 environment file `$SINAP/HOME/Bin/sinap_env[.csh or.sh]`. You do not need to assign a value to the variable. The SINAP/SS7 system simply verifies the existence of the variable.

Error Handling

When you develop an application, you typically expect some type of dialogue to occur between this application and one or more other applications. As you develop your application, you should be aware of all potential deviations from the planned dialogue and you should develop error-handling logic to process any and all of these deviations. For example, if your application expects to receive a RESPONSE from another application and instead receives a QUERY, your application should discard that MSU and return an error message.

NOTE

Upon receipt of an invalid component, your application should generate a `REJECT` message. It may also be appropriate for your application to deallocate any `T_Blocks` that had been allocated for the dialogue/transaction that failed.

The value returned by a CASL function indicates whether the function call was successful. To provide client application programmers with a familiar programming environment, CASL functions indicate an error condition by using the following UNIX-like methods.

- A function that normally returns a `0` or greater value will return `-1` if it is unsuccessful.
- A function that normally returns a pointer will return a `NULL` if it is unsuccessful.

When a CASL function fails, the function sets the variable `errno` to a specific error code to indicate the reason for the failure. The include file `$SINAP_HOME/Include/ca_error.h` defines the possible error codes a CASL function can return. The UNIX file `sys/errno.h` defines UNIX error codes and their meaning.

The global memory array `CA_ERR[]` contains an ASCII string in which the first several bytes are allocated for the error number, and the remainder of the array contains a description of the error. This array, which is defined in the `$SINAP_HOME/Include/sinapintf.h` include file, is not used by all CASL functions. Note, however, that all CASL functions return `errno`.

Error-Handling Considerations

The following list describes considerations of which you should be aware as you design and develop error-handling mechanisms for your application. The remainder of this section

describes the meaning of certain error-handling primitives your application can receive or be required to send, and provides instructions for how to handle these primitives.

- When a call to `ca_put_msg()` or `ca_put_msg_def()` fails, the SINAP/SS7 system calls the CASL function `ca_ipc_fail_event()` to inform trouble management of the failure. (An application process calls `ca_put_msg()` or `ca_put_msg_def()` to deliver an IPC message to another process.) The `ca_ipc_fail_event()` function delivers an alarm to trouble management to indicate that the SINAP/SS7 system was unable to deliver the IPC message. Upon error return, the application should attempt to resend the IPC message or it should perform appropriate error-handling actions.

  The alarm contains the following information.

  – The type of IPC message that could not be delivered, along with the process that sent the message (the source) and the process to which the message was destined (the destination).

  – The value of *`errno`* returned by the `ca_put_msg()` or `ca_put_msg_def()` function call that failed.

  – The number of times that `ca_ipc_fail_event()` was called by the process attempting to send the IPC message. (This count is important because critical alarms may be lost when a queue overflows.)

- When a call to `ca_get_msu()` or `ca_get_tc()` returns with a return code of `-1` and `errno` set to `EINTR`, the application process should read all of its IPC messages by calling `ca_get_msg()` in **nonblocking** mode (`fwait` set to `0`).

  Rather than indicating an actual error condition, this situation indicates that the application process received an IPC message while it was executing a blocking-mode call to `ca_get_msu()` or `ca_get_tc()`. To take advantage of this functionality without using signals, the application process must set its `register_req_t` structure's `fsignal` field to `2` (`IPC_NOTIFY_WITHOUT_SIGNAL`).

- The `IPC_NOTIFY_WITHOUT_SIGNAL` option for `fsignal` with `ca_register()` returns `EINTR` only if no MSUs are waiting and there is IPC. To avoid this problem, you can take one of the following actions in your application program:

  - Set `fsignal` to `TRUE`, have `SIG_S7_IPC` signal caught by the function that sets the global flag, and test this flag after each call to `ca_get_tc_ref()`. The `ca_get_tc_ref()` function passes its reference argument by value to the `get_msu_int(*prefwait)` function, thereby causing a window in the latter function before it tests this value, in which a signal (such as an IPC signal) could occur. In this case, the signal handler function's action of setting the global variable, `refwait`, to `FALSE` would have no effect and result in a blocking `read()` without a return of `EINTR`. This condition could produce a delay of IPC notification, which might be unacceptable for some applications.

- Call `ca_get_msg()` in non-blocking mode, looping until no IPC is available, before each call to `ca_get_tc()`. This eliminates both the need for IPC signals and the overhead of an additional system call each time.

Dialogue and Transaction Errors

When a dialogue/transaction error occurs, a single `tc_dhp_t` structure (CCITT, TTC, NTT, and China network variants) or `tc_thp_t` structure (ANSI network variant) is returned or sent. However, if the structure is being sent in response to an MSU that contained both dialogue/transaction and component parts, then multiple `T_Blocks` would have been assigned. In this case, it is the application's responsibility to deallocate those extra `T_Blocks`.

Table 3-34 describes the meaning of dialogue/transaction primitives and provides information about how your application should handle them.

Table 3-34. Dialogue/Transaction Primitives *(Page 1 of 2)*

Primitive	Description
TC_U_ABORT (Indication)	This primitive indicates that the other TCAP application has decided to abandon this transaction. The `dest_tid` field of the TC_U_ABORT request primitive identified the transaction being aborted. The reason for the abort is provided in the `pa_report_cause` field of the `tc_dhp_t` structure (CCITT/TTC/NTT/China) or the `tc_thp_t` structure (ANSI) that was returned with the primitive. If the structure's `tot_ua_info_length` field is greater than `0`, then more information is provided in the `ua_info` field. The application should call `ca_dealloc_tc()` to free up any component `T_Blocks` that were allocated for the dialogue/transaction. The application can also call `ca_put_event()` to log the error in order to trigger an event that results in operator intervention or causes the application to stop transaction processing.

Table 3-34. Dialogue/Transaction Primitives *(Page 2 of 2)*

Primitive	Description
TC_U_ABORT (Request)	This primitive indicates that this TCAP application has decided to abandon the current transaction due to a problem that makes normal response impossible. The application must do the following before issuing the primitive: • Fill in the `pa_report_cause`, `tot_ua_info_length`, and `ua_info` fields of the `tc_dhp_t` structure (CCITT/TTC/NTT/China) or the `tc_thp_t` structure (ANSI). • Swap the destination and originating dialogue/transaction IDs, which are defined in the `dest_tid` and `orig_tid` fields of the of the `tc_dhp_t` structure (CCITT/TTC/NTT/China) or the `tc_thp_t` structure (ANSI). • Call `ca_dealloc_tc()` to free up any component `T_Blocks` that were allocated for the dialogue/transaction. **Note:** The SINAP/SS7 system discards transaction components that had been previously sent by means of the `ca_put_tc()` function.
TC_P_ABORT (Indication)	This primitive indicates that the transaction was aborted by the SINAP/SS7 TCAP transaction-sublayer, perhaps because the transaction timer expired. (The timer's value is defined by the `register_req_t` structure's `tsl_timer_value` field.) The reason for the abort is provided in the `pa_report_cause` field of the `tc_dhp_t` structure (CCITT/TTC/NTT/China) or the `tc_thp_t` structure (ANSI) that was returned with the primitive. If the structure's `tot_ua_info_length` field is greater than `0`, then more information is provided in the `ua_info` field. The application should call `ca_dealloc_tc()` to free up any component `T_Blocks` that were allocated for the dialogue/transaction. The application can also call `ca_put_event()` to log the error in order to trigger an event that results in operator intervention or causes the application to stop transaction processing.

Component-Handling Errors

This section describes the primitives that an application may receive as the result of a component-handling error. These types of errors typically involve several component `T_Blocks`; therefore, the application should check the `last_comp_ind` field of the `tc_chp_t` structure to determine the exact number of `T_Blocks`. In addition, the application should also check to see whether it received a `NotLast` component flag.

Table 3-35 describes the meaning of component-handling primitives and provides information about how your application should handle them.

Table 3-35. Component-Handling Primitives *(Page 1 of 5)*

Primitive	Description
TC_L_CANCEL (Indication)	The SINAP/SS7 system issues this primitive to indicate that the transaction was aborted because the transaction timed out. (The length of time allowed to process the transaction was specified by the calling party in the `tc_chp_t` structure's `timer_value` field.) Only a single component is returned (in the `tc_chp_t` structure); the dialogue/transaction component is not returned. The SINAP/SS7 system issues a `TC_L_CANCEL` primitive for each of the transaction's INVOKE primitives. The application should call `ca_dealloc_tc()` to free up any component `T_Blocks` that were allocated for the dialogue/transaction. The application can also call `ca_put_event()` to log the error in order to trigger an event that results in operator intervention or causes the application to stop transaction processing. **Note:** If the transaction responds at a later time, the result will be discarded by the SINAP/SS7 TCAP transaction sublayer (TSL).
TC_U_CANCEL (Request)	Your application can issue this primitive to discard any components that have already been sent to TCAP's component sublayer for processing. Upon receipt of this primitive, the SINAP/SS7 system cancels the sending of pending components. The application must issue this primitive before calling the `ca_put_tc()` function to send the dialogue/transaction component for this transaction (identified by the `trans_id` field of the `t_block_t` structure). Once the application issues the `TC_U_CANCEL` request primitive, it can resume sending new components for the transaction.

Table 3-35. Component-Handling Primitives *(Page 2 of 5)*

Primitive	Description
TC_U_ERROR (Request)	The other TCAP application issues this primitive to indicate that it was unable to execute the requested operation. Upon receipt of this primitive, your application should evaluate the `tc_chp_t` structure's data field to determine the cause of the problem; the other application will have filled in the Error Code tag value based on the definitions in ITU-T (CCITT) Recommendation Q.773, Table 25, or ANSI Recommendation T1.114.3. In addition, this field will contain application-specific information values for Error Code length and data. The other application may also return optional tagged parameters that are application specific.
TC_U_ERROR (Indication)	The other TCAP application issues this primitive to indicate that an operational error occurred. Upon receipt of this primitive, your application should perform whatever action has been agreed upon beforehand and should deallocate this `T_Block` component. If there was only a single component, your application should also deallocate the dialogue/transaction `T_Block` component. In addition, you may want your application to call `ca_put_event()` to log the error, which is useful for triggering an event that results in operator intervention or terminates the application.
TC_U_REJECT (Request)	Your application can issue this primitive to indicate an improperly constructed component. The problem is typically an unknown parameter ID. The application issuing this primitive fills in the `tc_chp_t` structure's `problem_type` and `problem_specifier` fields with standard values to define the problem. In addition, the application can also provide optional parameters in the `tc_chp_t` structure's `data` field. **Note:** An application issuing this primitive must issue a separate primitive for each of the MSU's components, each of which has the same invoke ID (specified in the `tc_chp_t` structure's `invoke_id` field).

Table 3-35. Component-Handling Primitives *(Page 3 of 5)*

Primitive	Description
TC_U_REJECT (Indication)	The other TCAP application issues this primitive to indicate that it received an improperly constructed component in a TC_U_REJECT request primitive. Upon receipt of this primitive, your application should examine the tc_chp_t structure's problem_type and problem_specifier fields to determine what caused the problem. Your application should then perform the appropriate steps to handle the situation. In addition, you may want your application to call ca_put_event() to log the error. This is useful for triggering an event that results in operator intervention or terminates the application.

Table 3-35. Component-Handling Primitives *(Page 4 of 5)*

Primitive	Description
TC_L_REJECT (Indication)	The SINAP/SS7 component sublayer issues this primitive and sends it to the local TCAP user to indicate receipt of a duplicate or invalid component. The component sublayer moves the information in the problem_code field for the invalid MSU into the problem_type and problem_specifier fields of the tc_chp_t structure. In addition, the SINAP/SS7 Invoke State Mechanism (ISM) generates a TC_R_REJECT indication primitive, which it will send to the remote TCAP user if the local TCAP user decides to process the invalid MSU and continue with the dialogue/transaction. The TC_R_REJECT primitive will not be sent if the error occurs during an ending operation such as TC_RESULT_LAST. Upon receipt of this primitive, your application can do either of the following: • *Terminate the dialogue/transaction*. To do this, your application should format a TC_END message (CCITT) or a TC_RESPONSE message (ANSI). The SINAP/SS7 system will automatically format a REJECT component using information from the ISM. In addition, your application should deallocate any T_Block components allocated for the dialogue/transaction. • *Continue the dialogue/transaction with a rejected component*. If you want to continue a dialogue/transaction even though one of its components has been rejected, your application should perform whatever procedures it and the other TCAP user previously agreed upon. When your application issues a call to ca_put_tc() to send continuation data for the dialogue/transaction, the SINAP/SS7 system will send the data and the TC_R_REJECT primitive to the other TCAP user. You might also want your application to call ca_put_event() to log the error. This is useful for triggering an event that results in operator intervention or terminates the application.

Table 3-35. Component-Handling Primitives *(Page 5 of 5)*

Primitive	Description
TC_R_REJECT (Indication)	This primitive indicates that the remote TCAP user received an improperly constructed MSU but has decided to continue the dialogue/transaction anyway. The information in the invalid MSU's problem_code field is written to the tc_chp_t structure's problem_type and problem_specifier fields. Typically, TC_R_REJECT primitives indicate receipt of a reject MSU. A TC_R_REJECT T_Block is generated for all types of general problems: an unrecognized correlation ID for an INVOKE problem type, an unrecognized correlation ID and an unexpected RR for a RETURN RESULT problem type, and an unrecognized correlation ID and unexpected RE for a RETURN ERROR problem type. Upon receipt of this primitive, your application should do the following: • Examine the problem_type and problem_specifier fields of the tc_chp_t structure to determine why the MSU was improperly constructed. • Deallocate any T_Block components allocated for the dialogue/transaction. • Perform whatever steps were agreed upon with the other TCAP user. • Optionally, you may want your application to call ca_put_event( ) to log the error. This is useful for triggering an event that results in operator intervention or terminates the application.

Triggering Events and Trouble Treatment

Trouble Management handles events according to information in the trouble treatment table (treat.tab). You create the basis for this table by editing the file $SINAP_HOME/Library/treat.tab.

Adding an Event or Changing Its Treatment

The following steps show how to add an event or change the way the SINAP/SS7 system treats it.

1. Edit the treat.tab file in $SINAP_HOME/Library to add an event or change the way the SINAP/SS7 system treats it.

2. Use the `nmtr` program to convert `treat.tab` to an intermediate format called `tm_treat.lod`. The `nmtr` program has the following syntax (where *`filename`* is the name of the file to be converted).

   ```
   nmtr [-I] filename
   ```

   The `-I` argument installs the `tm_treat.lod` file in the directory `$SINAP_HOME/Bin/shm/pri` and renames the file to `TREAT_load`. If you do not use this argument, you must copy the file manually to the directory and rename it `TREAT_load`.

3. Invoke the SINAP/SS7 MML command `READ-TREAT`. The SINAP/SS7 system reads the `TREAT_load` file into the trouble treatment table.

   NOTE

   When you invoke `READ-TREAT`, the SINAP/SS7 system reinitializes any counters (that is, accumulation of events per time period) to 0.

Setting Up the Trouble Treatment Table

The following section describes the format of the trouble treatment table (`treat.tab`). When you edit `treat.tab`, you designate the category and subcategory treatment that Trouble Management implements when it receives the event.

The `treat.tab` file has the following format.

```
TREAT_BEGIN
CATEGORY=category
 SUBCATEGORY=subcategory
 FREQ_COUNT=freq_count
  FREQ_WINDOW=freq_window
  ALARM_TYPE=alarm_type
  ALARM_COLOR=alarm_color
  ALARM_TRIGGER=alarm_trigger
  TERM_PROCESS_TRIGGER=term_process_trigger
  TERM_SUBSYS_TRIGGER=term_subsys_trigger
  SCRIPT_PATH=script_path
  SCRIPT_TRIGGER=script_trigger
  NEW_EVENT=new_category,new_subcategory
  NEW_EVENT_TRIGGER=new_event_trigger
 SUBCAT_END
 SUBCATEGORY=subcategory
     .
     .
     .
 SUBCAT_END
CAT_END
CATEGORY=category
 SUBCATEGORY=subcategory
     .
     .
     .
TREAT_END
```

Table 3-36 lists and describes the fields in the `treat.tab` file.

Table 3-36. Trouble Treatment Table (`treat.tab`) Fields *(Page 1 of 3)*

Field Name	Description
CATEGORY	Specifies the alarm category. Categories are defined system wide; each value for the CATEGORY field has a unique meaning to the system. The value you assign to CATEGORY can be any number from 1 through 30 or a label. A label is a constant (#define value) that you create in your own include file. The trouble treatment table can contain multiple categories. However, you must specify each category in ascending order; for example, category 2 cannot precede category 1.

Table 3-36. Trouble Treatment Table (`treat.tab`) Fields *(Page 2 of 3)*

Field Name	Description
SUBCATEGORY	Specifies the classification of an event within a category. The value you assign to SUBCATEGORY can range from 1 through 30 or be a label. There can be multiple subcategories in the file. You must specify each subcategory in ascending order; for example, subcategory 2 cannot precede subcategory 1.
FREQ_COUNT	Specifies the count of events to occur within the time specified by FREQ_WINDOW before the SINAP/SS7 system triggers an alarm. You must assign either an integer value or a label to FREQ_COUNT.
FREQ_WINDOW	Specifies the interval, in seconds, in which to count events before the SINAP/SS7 system triggers an event. You must assign either an integer value or a label to FREQ_WINDOW.
ALARM_TYPE	Specifies the type of alarm the SINAP/SS7 system will send to all processes that are registered to receive this alarm class. Possible values are REG_CRITICAL, REG_MAJOR, REG_MINOR, or REG_NOTICE. Critical alarms are severe, service-affecting conditions that require immediate attention. These alarms are automatically written to the UNIX system log and system console. Major alarms are the result of hardware or software conditions that indicate a serious disruption of service. Minor alarms are the result of troubles that do not have a serious effect on customer service. Notice alarms are the result of a service-affecting situation and are provided for information purposes only.
ALARM_COLOR	Specifies the severity of the alarm. Possible values are RED, YELLOW, GREEN, and WHITE, which correspond to the numbers displayed in the alarm (4 through 1, respectively). Currently, the SINAP/SS7 system does not use this field.
ALARM_TRIGGER	Specifies the type of alarm to trigger. Possible values are: • SINGLE_EVENT triggers an alarm on a single event. • FREQ_MON triggers an alarm when the frequency count is exceeded within the time specified by FREQ_WINDOW.
TERM_PROCESS_ TRIGGER	Specifies when to terminate the process that has sent the alarm. Possible values are SINGLE_EVENT and FREQ_MON.
TERM_SUBSYS_ TRIGGER	Specifies when to terminate the subsystem with the offending process. Possible values are SINGLE_EVENT and FREQ_MON.

Table 3-36. Trouble Treatment Table (`treat.tab`) Fields *(Page 3 of 3)*

Field Name	Description
SCRIPT_PATH	Specifies the path name of a user-defined script file. Trouble Management spawns a shell to call a script you have defined. You can use scripts for any purpose you define; however, you should not program real-time actions using the script. The name of the script can be up to 31 characters long.
SCRIPT_TRIGGER	Specifies when to trigger the script. Possible values are SINGLE_EVENT and FREQ_MON.
NEW_EVENT	Specifies the category and subcategory for the new event the SINAP/SS7 system generates upon receiving the specified trigger. You can specify a new event with a new category and subcategory. The new event contains the ASCII string from the original event and goes to Trouble Management. Trouble Management limits the daisy chaining of events to a predefined limit of 32. This limit is imposed to prevent event messages from circulating in an endless loop.
NEW_EVENT_ TRIGGER	Specifies when to trigger the new event. Possible values are SINGLE_EVENT and FREQ_MON.

Specify comments in the treat.tab file by enclosing them with a slash and an asterisk, for example, `/* information */`.

Within a subcategory definition, you must combine certain actions. For example, to specify frequency monitoring, you must use the FREQ_COUNT and FREQ_WINDOW fields. To specify alarms, you must use the ALARM_TYPE, ALARM_COLOR, and ALARM_TRIGGER fields. To specify script files, you must use both the SCRIPT_PATH and SCRIPT_TRIGGER fields. To specify secondary events, you must use both the NEW_EVENT and NEW_EVENT_TRIGGER fields.

To disable any action, do not specify it or any of its associated actions.

Figure 3-18 shows an example of the file used to create a trouble treatment table.

```
#include <caslinc.h>       /*SINAP Include Files */
#include <dr_incl.h>      /* SINAP Driver Include File */

   /* User defined include files should be included here */
  /* Explicit path names are required for files not in */
 /* /home/sinap/Include */

/* Define treatment for Client Application Z (assigned number 23) */

TREAT_BEGIN
 CATEGORY=CLIENT_Z

/* Subcategory */

SUBCATEGORY=CLIENTZ_SUB1

     /* Generate trigger after 3 events in 10 seconds */
        FREQ_COUNT=3
        FREQ_WINDOW=10

     /* Trigger Minor Alarm after a Single Event */
        ALARM_TYPE=REG_MINOR
        ALARM_COLOR=YELLOW
        ALARM_TRIGGER=SINGLE_EVENT

    / * Generate New Event (CLIENT_Z, CLIENTZ_SUB2) Upon Frequency Mon.Trigger */
        NEW_EVENT=CLIENT_Z, CLIENTZ_SUB2
        NEW_EVENT_TRIGGER=FREQ_MON

   SUBCAT_END

   /* Subcategory CLIENTZ_SUB2 */

SUBCATEGORY=CLIENTZ_SUB2

   /* Trigger Critical Alarm after a Single Event */
      ALARM_TYPE=REG_CRITICAL
      ALARM_COLOR=RED
      ALARM_TRIGGER=SINGLE_EVENT
```

Figure 3-18. Sample `treat.tab` File

Figure 3-18. (Continued) Sample `treat.tab` File

```
   /* Terminate Subsystem */

      TERM_SUBSYS_TRIGGER=SINGLE_EVENT

   /* Execute User Script */

      SCRIPT_PATH="/usr/client/zdir/restartz"
      SCRIPT_TRIGGER=SINGLE_EVENT

     SUBCAT_END
 CAT_END
TREAT_END
```

Chapter 4
Application Testing, Debugging, and Troubleshooting

This chapter provides information about testing and troubleshooting applications and debugging them if necessary. It contains the following sections.

- “Listing Active SINAP/SS7 Processes” provides instructions for determining whether all SINAP/SS7 processes are active, which is the first thing you should do if you suspect a problem with your client application.
- “Evaluating Alarms and Events” describes the types of alarms and events that the SINAP/SS7 system reports. These alarms and events can sometimes aid you in debugging a problem with your client application.
- “The BITE Subsystem” describes how you can use the BITE subsystem to troubleshoot and debug your client application. BITE consists of several facilities for monitoring and evaluating the operation of a client application: scenario execution, a monitor facility, a log-analysis program, and an MML command for sending debug messages to an application.
- “BITE Commands Reference” provides a description of the MML commands that you can use to monitor, debug, and troubleshoot a client application.
- “Measurement Collection Commands” provides a description of several MML commands that you can use to collect measurements related to the number and types of messages that the SINAP/SS7 system sends and receives.
- “Log-Analysis Commands Reference”describes the log-analysis commands and their relational operators.

Listing Active SINAP/SS7 Processes

If you suspect you have a problem with one of your client applications, first determine if all the SINAP/SS7 processes are running. Each SINAP/SS7 process has a unique label. Table 4-1 contains an alphabetical listing of SINAP/SS7 processes and their labels.

To determine whether a particular SINAP/SS7 process is active, you must display a list of active system processes, then check the command's output to see if any SINAP/SS7 processes are listed. To display a full listing of all active processes on the system, issue the following command from the UNIX prompt. From the command output, you can determine which SINAP/SS7 processes are currently active.

```
ps -eaf | grep sinap | more
```

Table 4-1. SINAP/SS7 Process Labels *(Page 1 of 2)*

BITE Subsystem Processes	
`bibp`	BITE Parent Process
`bilf`	Log File Process
`bimi`	Man Machine Interface
`bitu`	Link Test User Part Process
MTP Subsystem Processes	
`l3cb`	Changeback Process
`l3co`	Changeover Process
`l3cr`	Controlled Rerouting Process
`l3dt`	Message Distribution Process
`l3fr`	Forced Rerouting Process
`l3la`	Link Availability Control Process
`l3ls`	Link Set Control Process
`l3mp`	MTP Management Parent Process
`l3mt`	MTP Level-3 Management Process
`l3rc`	signaling Routing Process
`l3rt`	Message Routing Function
Node Management Processes	
`nmc1`	Client Management Process

Table 4-1. SINAP/SS7 Process Labels *(Page 2 of 2)*

`nmcm`	Command Management Process
`nmdm`	Deferred Message Process
`nmds`	Disk I/O Server Process
`nmip`	IPC Handler Process
`nmmc`	Measurement Collection Process
`nmni`	SS7 Network Interface Process
`nmnp`	Node Management Parent Process
`nmth`	Terminal Handler Process
`nmtm`	Trouble Management Process
SCCP Management Process	
`scmg`	SCCP Management Process

Evaluating Alarms and Events

The SINAP/SS7 system produces alarms and events. An *event* is an unusual occurrence that may or may not result in an error. Trouble Management logs every software event in the Software Notebook and every hardware or network event in the Alarm History log. The category and subcategory of the event (specified with the `ca_put_event()` function call) determine the file to which an event is logged. An *alarm* is an error message that results from a particular command or system event. Alarms are identified according to the part of the SINAP/SS7 system from which they originate, and appear only on terminals registered for alarm display.

An alarm consists of the following components.

- Originator information specifies the entity that detected the alarm. In Figure 4-1, the first line contains the data `N1, M1, NM, TM`. This indicates that the process on the default node (`N1`), default module (`M1`), of Trouble Management (`TM`) within Node Management (`NM`) detected the alarm.
- Alarm information gives the date and time of the alarm, its class, severity, and color. In Figure 4-1, the first alarm has a class of 4. It is a critical alarm requiring immediate action, and its color is 4 or red. (See ‘‘Setting Up the Trouble Treatment Table’’ in Chapter 3 for information about alarm classes and colors.)
- Alarm data in the form of an ASCII string explains why the alarm was generated. In Figure 4-1, the last line of the second alarm contains the line `Health check timeout, pid = 471`, indicating that the alarm was sent because of a health check timeout for process ID 471.

Figure 4-1 shows two sample alarms.

```
N1,M1,NM,TM, 1998-06-29 11:46:25,
Class = 4, Critical Alarm - Immediate Action Required, Color = 4
**** Process NM,NI (pid=5667) died..., signal=9

Trouble Notification:
N1,M1,NM,TM, 1998-07-12 21:57:11,
Class = 2, Minor Alarm - Repairs Required, Color = 2
Health check timeout, pid = 471
```

Figure 4-1. Sample Alarm Format

Alarm Notification and Severity

All processes that are registered to receive a specific alarm category and type receive alarm notification. By default, SINAP/SS7 alarm messages are written to the following file (where *mmdd* is the date).

`$SINAP_HOME/Logs/system/ALM`*mmdd*

Alarms and Software Notebook Events

To determine which subsystem is reporting an alarm, check `errno` for an error message code and an accompanying error message. The following is a list of the `errno` values used by specific SINAP/SS7 subsystems. These values are defined in the SINAP/SS7 `ca_error.h` include file.

***errno* Value**	**Subsystem**
1 – 256	UNIX or SS7 Driver
1000 – 1999	Node Management
2000 – 2999	CASL
3000 – 3999	TCAP
4000 – 4999	SCCP
5000 – 5999	MTP
6000 – 6999	BITE
7000 – 7999	Client application

Software Notebook Events and Messages

The Node Management Software Notebook is a log of software events and error messages which you can access by using the MML command `REPORT-NBOOK`.

MTP Alarms

MTP alarms can take two forms. For a description of these forms and an explanation of the related events, see the include file `$SINAP_HOME/Include/mtpevents.h`.

MTP level-3 Management Process errors have the following format.

```
l3mt module #, state # - input # error,
```

All other MTP level-3 processes have the following format.

```
MTP3 state event: sub: #, state: #, code: 0x#
```

Nondata Primitives

If the SCCP or TCAP sends a nondata primitive, use the appropriate standards documentation to determine the problem and decode the messages.

System Log File

The SINAP/SS7 system writes some of its messages to the UNIX system log file. While some of the messages are normal, others indicate a potential problem with the SINAP/SS7 system or UNIX. See the *SINAP/SS7 User's Guide* (R8051) for information about the types of messages that the SINAP/SS7 system writes to the UNIX system log file.

User-Supplied Error Messages and Events

A client application process can send events and error messages as IPC messages to Node Management's Trouble Management Process. Upon receiving an event, Trouble Management logs the event to disk and determines how to respond to it, based on the event category and subcategory specified in the event. You set the treatment for a particular category and subcategory combination in the trouble treatment table.

Because all user events are software events, they are logged in the Software Notebook. You can program events for all categories and subcategories. The only exceptions are the hardcoded alarms that are generated when you have not specified treatment for a particular category and subcategory in an event, or when event daisy chaining exceeds the predefined limit.

The BITE Subsystem

The BITE subsystem provides built-in and external test and monitoring facilities to help you detect problem areas, monitor the traffic on various links, and simulate network operation in order to test applications without affecting the normal processing of your SINAP/SS7 configuration. BITE provides the following capabilities, each of which is described in the sections that follow.

- The *monitor facility* enables you to monitor applications, processes, SS7 links, and IPC paths. During a monitor session, information is collected and written to a BITE log file, which you can then analyze through BITE's log-analysis program. You use the MML commands and , respectively, to initiate and terminate a BITE monitor process. For more information, see "The BITE Monitor Facility" and the MML command descriptions later in this chapter.
- The *log-analysis* program processes information logged by BITE. This program provides several commands for finding, displaying, and selecting particular records in a log file, and for printing the results. For more information, see "The BITE Log-Analysis Program" later in this chapter.
- The *scenario execution* feature provides a controlled environment in which to simulate network operation for the purpose of testing a SINAP/SS7 application. You can determine whether the application is functioning correctly by monitoring its operation during the scenario execution. You use the MML commands and , respectively, to start and stop a scenario execution. (These commands are described in detail later in this chapter.)

  In addition, the *Database Builder* program constructs the MSU(s) to be used with scenario execution. For more information about scenario execution or the database builder program, see the "Scenario Execution" section later in this chapter.

- The MML command enables you to send debug messages to an active application. (See its command description later in this chapter.)

The BITE Monitor Facility

The BITE monitor facility traces messages from any level of the system, and thus provides an in-depth look at the system's status. This means that you can check an application's status at the SS7 level. For example, suppose you confirm that your SINAP/SS7 configuration is functioning, but MSUs are not being processed. You can use the BITE to further investigate the problem and to determine whether the application is receiving and responding to inquiries and whether the SS7 driver is sending information.

You initiate a BITE monitor process by issuing the MML command `START-MON`, specifying the entities to be monitored and the types of operations for which you want to collect information (read or write, or both). The monitor process keeps track of the specified entities in order to collect the specified information, which it writes to a BITE log. You must issue the MML `STOP-MON` command to terminate the monitoring session. Then, you can use BITE

log-analysis commands to display and extract records and obtain summary information from the BITE log, which contains the data collected during the monitoring session.

A BITE log contains all records received during a BITE monitoring session, in the order in which they are received. These records are in `I_Block` format, with SS7 `M_Block` structures embedded in the `I_Block` structure. BITE monitor logs can contain `IPC`, `SS7`, and `LNK` messages.

- If the log contains IPC messages, see the `iblock.h` include file to decode the message type. You can then use the appropriate `.h` file to decode the message structure.
- If the log contains `SS7` or `LNK` messages, use the `mblock.h` include file to determine the message type and structure. You may also need to refer to the appropriate standards documentation to decode messages. However, the log-analysis program performs most of this decoding for you.

Scenario Execution

Scenario execution is a BITE feature that provides a controlled environment in which you can test a client application to see how it operates. During a scenario execution, you simulate the operation of an SS7 network by running two applications simultaneously: one is the application being tested (the *test application*) and the other is a specialized s*cenario-execution application* that acts as a peer to the test application.

To evaluate the test application's operation, you can use the BITE facility to monitor communication between the test and scenario-execution applications during the scenario execution. To do this, initiate a BITE logging session in another window by issuing the MML command. The BITE facility logs communication between both applications to a log file. When the scenario has finished executing (when the scenario-execution application has terminated), you can use the BITE log-analysis commands to examine the log file and determine whether the test application is operating correctly. In this way, you can test your client application (the test application) without running it in an SS7 network.

The Scenario-Execution Application (`se_send`)

The scenario-execution application acts as the test application's peer: it sends messages to the test application and processes any responses. Thus, the scenario-execution application simulates the operation of an application with which the test application would communicate. For example, if you are testing an MTP application, you would develop a scenario-execution application that transfers MSUs from the test application and accepts responses from the test application.

A sample scenario-execution application is included with the SINAP/SS7 software, `$SINAP_HOME/Samples/se_send`. You can use the `se_send` application to test client applications that you have developed or you can use it as a template for developing other scenario-execution applications.

The `se_send` program does the following:

- Sends TCAP messages (like BEGIN and INVOKE) to the application being tested
- Receives responses (like END/RETURN) from the application being tested
- Works with applications registered at any SS7 protocol level (MTP, SCCP, and TCAP)

NOTE

If you want to modify the `se_send` application, Stratus recommends making a copy of `se_send` and modifying the copy.

Using the Database Builder to Create Test MSUs

To run scenario execution, you must construct a test MSU for the scenario-execution application to send to the test application. You **must** use the Database Builder program to construct this MSU. The Database Builder is a menu-driven interface which can be used to build different types of MSU messages for different types of applications and scenarios. When you construct a test MSU, you must define the MSUs MTP routing label, the SCCP message (which includes message type, protocol class, called- and calling-party addresses), and the TCAP data. The Database Builder program automatically constructs the MTP header and its control structure. In the MTP header's control structure, the message ID and size are set; all other fields are initialized to `0`.

To create a test MSU for a scenario execution, perform the following steps.

1. Use either of the following methods to create an ASCII file that contains the TCAP data you want to include in the MSU. This is the file whose path name you will specify for option `20` (`TCAP data`) of the Database Builder menu.
   - Create the file by using a standard text editor such as `vi`. Format the TCAP user data according to the standards documentation appropriate for the variant of the SINAP/SS7 system you are using. Data should appear as character pairs, separated by spaces, dashes, horizontal tabs, carriage returns, or line feeds.

     NOTE

     You **must** use hexadecimal format for the data.

     The following example shows a sample line of data in the TCAP data file.

     ```
     60 61 5E 6C A1 2B 02 01 02 06 02 83 01 F2 22
     ```

   - Use the TCAP data in a BITE log. You can do this by saving the BITE log to a system file and then using a standard text editor (such as `vi`) to extract the TCAP data and write it to a separate file. (For information about how to save a BITE log to a system file, see the description of the log-analysis command later in this chapter.)

2. Activate the Database Builder program by typing the following command from the command line of a SINAP/SS7 login window. (The *message_file* argument is an optional argument that specifies the path name of a file to which an existing test MSU was saved. To open the file and display that MSU, include this argument in the command line.)

   ```
   bidb [message_file]
   ```

   The following Database Builder menu displays.

   NOTE

   The values shown are the default values for the menu options. If you opened an existing test MSU file, the values for that MSU would be displayed instead.

```
X - Exit and Create file; Q - Quit
1.  SIO: 3
2.  DPC: 2730
3.  OPC: 0
4.  SLS: 1
10. SCCP msg type       : UNITDATA
11. SCCP protocol class : 0, no return on error
12. SCCP called address : pc(none) ssn(none)
13. SCCP calling address: pc(none) ssn(none)
20. TCAP data:


enter option>
```

Figure 4-2. The Database Builder Menu

3. Construct a test MSU by specifying an appropriate value for each of the Database Builder menu options. If the option's default value is appropriate, you need not specify a value for that option.

   To select a Database Builder menu option, use the keyboard to type in the number that corresponds to that option and press **RETURN**. In most cases, the system will display a prompt that contains a list of valid values for that option. Use the keyboard to type in the desired value or its corresponding number and press **RETURN**. The Database Builder displays the value you specified for that menu option.

   The Database Builder menu options are described here.

- Option `1` specifies the service information octet (SIO). Valid values are in the range `1` through `15`.
- Options `2` and `3` specify the MSU's destination point code (DPC) and the originating point code (OPC), respectively. Specify the point code by using the appropriate address format for the variant of the SINAP/SS7 system you are using (for example, `2730` for CCITT or `254-12-8` for ANSI.)

  NOTE

  If you plan to run the scenario-execution application and the test application on the same system, specify that system's point code for both the DPC and the OPC.

- Option `4` specifies the signaling link selection (SLS) field. Valid values are in the range `0` to `15`.
- Option `10` specifies the type of SCCP message you are constructing (`9` – `UNITDATA` or `10` – `UNITDATA_SVC`).
- Option `11` specifies the type of SCCP protocol class (`0` or `1`) and the error return option (`0` – `NO RETURN ON ERROR` or `8` – `RETURN ON ERROR`).
- Option `12` specifies the SCCP called-party address (`pc` and `ssn`)
- Option `13` specifies the SCCP calling-party address (`pc` and `ssn`).
- Option `20` specifies the path name of an ASCII file that contains the TCAP data to be included in the message. For information about how to create this file, see Step 1.

4. When you have specified a value for all of the menu options, enter the value `X`. The program prompts for the name of a file to which to save the test MSU. Provide the requested information and press **RETURN**.

   The Database Builder program constructs the test MSU and saves it to the specified file. (This file is then used as input to the `START-SCEN` command, which initiates a scenario execution.) The program exits and you are returned to the command line of the SINAP/SS7 login window from which you invoked the Database Builder program.

5. Proceed to the following section, "Procedures for Running a Scenario Execution," for instructions on running the scenario execution.

Procedures for Running a Scenario Execution

Once you have created the test MSU to be used for your scenario execution session, perform the following steps to initiate the scenario execution.

1. Activate the application you plan to test. For example, if you are testing the sample TCAP program, `tcrecv.c`, you would issue the command `tcrecv` to activate the application.

NOTE

When you built the test MSU in the preceding procedure, you should have specified the SSN of the test application as the value of the SCCP called-party address, which appears as option `12` in the Database Builder menu.

2. Open two windows on the same terminal or on two different terminals. Use the appropriate procedure for the window manager you are using on your system. (For instructions, see the documentation for that window manager.)
3. In one window, activate the scenario-execution application by issuing the following command. For the `FILE` parameter, specify the path name of the scenario-execution application; in this case, `se_send`.

```
START-SCEN:ENT=(,,TCAP,RECV),FILE=$SINAP_HOME/se_send;
```

NOTE

You must have already activated the application being tested (step 1); otherwise, the scenario-execution application will send MSUs to an SSN that is unavailable and the scenario execution will fail.

4. In the other window, or on another terminal, issue the following command to start a BITE monitoring session (where *`log`* is the name of the log file to which to write the results). You use the BITE facility to monitor and log transactions between the scenario execution application (in this case `se_send.c`) and the application being tested (in this case `tcrecv.c`).

```
START-MON:ENT=(SS7,,TCAP,RECV,),DISP=Y,LOG=log;
```

5. When the scenario execution has finished executing, issue the following command to obtain the ID number of the scenario, which you will need to halt it. Issue this command from the same window in which the scenario execution was running.

```
DISPLAY-SCEN;
```

6. Issue the following command to halt the scenario execution (where *`scenario_id`* is the ID that the SINAP/SS7 system assigned to this scenario-execution process).

```
STOP-SCEN:ENT=scenario_id; <
```

The BITE Log-Analysis Program

You can use the BITE log-analysis program to display and analyze information in a BITE log. You can invoke this program by performing either of the following actions.

- Type `bila` from the command line of a SINAP/SS7 login window (that is, any window through which you have logged in as the user `sinap`).
- Select the `Log Analysis` option from the Terminal Handler's BITE menu.

The SINAP/SS7 system activates the log-analysis program and displays the log-analysis menu, shown in the following panel.

```
********************* BITE   LOG   ANALYSIS *********************
The Syntax of the MML Command is:

COMMAND:PARAMETER=[OPERATION]VALUE,...;

     The Log Analysis Commands are:

        – FIND

        – SELECT

        – SUMMARY

        – DISPLAY

        – QUIT


Enter the Log Analysis MML Command, or enter 'help' for more information:
```

To issue a BITE log-analysis command, use the keyboard to type in the command line for the command; or, type `help` to display help information. (The log-analysis commands are described in the section "Log-Analysis Commands Reference" later in this chapter.)

Be aware of the following considerations as you issue log-analysis commands.

- You must enter the complete command line for the command you want to execute (for example, `display:file=log_0529` and not simply `display`). Unlike the Terminal Handler, the BITE log-analysis program does not build a command based on your input to specific prompts. (For a description of command's syntax, see the command description later in this chapter.)
- You can type the command in lowercase letters (for example, `find:file=log0529`); you need not use uppercase letters (for example, `FIND:FILE=LOG0529`).
- The semi-colon (`;`) at the end of the command is optional; you need not include it in the command line.

Log-Analysis Commands Reference

This section describes the following log-analysis commands:

- DISPLAY
- FIND
- SELECT
- SUMMARY
- QUIT

Each command accepts a relational operator for any of its arguments. For example, in the *key_value* argument of the FIND command, you can use the value 2&&4. This indicates that all records for OPC 2, 3, and 4 are to be extracted. In addition, you must use these relational operators when specifying the *key_value* argument. For descriptions of these commands and keywords, see the following two tables.

Table 4-2 shows a list of available relational operators.

Table 4-2. Relational Operators

Operator	Notation
Equal To	=
Not Equal To	~=
Greater Than	>
Less Than	<
Greater Than or Equal To	>=
Less Than or Equal To	<=
Range	&&
A Set of Specific Numbers	|

The = and ~= operators compare numeric values or ASCII strings.

NOTE

ASCII strings must be enclosed in quotation marks (").

The >=, <=, <, and > operators compare numeric values. The && operator compares an inclusive range of two numeric values. The | operator compares a set of specific numeric values.

To locate particular records in a log file, you can include one of the keywords listed in the following table in the log-analysis command. The log-analysis program searches the log file for records that match the specified criteria. For example, to find all records in the log file `LOGSS710` that begin at 12:00, end at 1:30, and contain an SIO of `3`, include the `FILE`, `BT`, `ET`, and `SIO` keywords in the `FIND` command line as follows:

```
FIND:FILE=LOGSS710 BT=12:00:00,ET=1:30:00,SIO=3;
```

Table 4-3 lists the keywords available for searching for a record in a log file.

Table 4-3. Keywords for Searching Log File Records *(Page 1 of 2)*

Keyword	Meaning
General Keywords (for searching all records)	
BT	Begin Time (use last I_Block time stamp)
ET	End Time (use last I_Block time stamp)
FILE	Log File Name
OFILE	Output File Name
I_Block Keywords (for searching I_Block records)	
MSG	I_Block Message Type
REF	I_Block Reference Number
MON	I_Block Monitor ID
SCEN	I_Block Scenario ID
O_ENT	I_Block Originator Entity – Specify as (*NNAME*,*MNAME*,*ANAME*,*PNAME*,*INST*)
D_ENT	I_Block Destination Entity – Specify as (*NNAME*,*MNAME*,*ANAME*,*PNAME*,*INST*)
M_Block Keywords (for searching M_Block records)	
TIME	M_Block Time Stamp
CA_LNK	M_Block CASL Control Physical Link ID
CA_SRC	M_Block CASL Control Source Field
CA_SND	M_Block CASL Control Message Sender
CA_CNT	M_Block CASL Control Lost M_Block Count
SC_MSG	M_Block SCCP Message Type

Table 4-3. Keywords for Searching Log File Records *(Page 2 of 2)*

Keyword	Meaning
SC_PRO	M_Block SCCP Protocol
MT_MSG	M_Block MTP Control Message ID
MT_SEQ	M_Block MTP Control Sequence Number
MT_SID	M_Block MTP Sender ID
MT_SZ	M_Block MTP Message Size
BIB	M_Block L2 BIB-BSN
FIB	M_Block L2 FIB-FSN
LI	M_Block L2 LI
SIO	M_Block SIO
DPC	M_Block DPC
OPC	M_Block OPC
SLS	M_Block SLS
HO1	M_Block H0-H1 Code

DISPLAY

SYNOPSIS

`DISPLAY:FILE=`*`logfile`*`;`

DESCRIPTION

The `DISPLAY` command displays the BITE log specified by the *`logfile`* argument. Include the log file's complete path name in the *`logfile`* argument; otherwise, the command returns an error.

The command displays each record's field, starting with the `I_Block` header, and followed by the data contents. If the record is an IPC message, the data is displayed in hexadecimal format. If the record data is an SS7 message (`M_Block`), then the display is further broken down to the `L2`, `SIO`, `DPC`, `OPC`, `SLS`, and `MTP` message types, and hexadecimal data dumps (SCCP and TCAP message contents).

- If the log contains IPC messages, see the `iblock.h` include file to decode the message type. You can then use the appropriate `.h` file to decode the message structure.
- If the log contains `SS7` or `LNK` messages, use the `mblock.h` include file to determine the message type and structure. You may also need to refer to the appropriate standards documentation to decode messages; however, the log-analysis program performs most of this decoding for you.

Once the file displays in the log-analysis program, you can save it to a system file by entering the following command at the log-analysis prompt (`:`). For *`filename`*, specify the path name of the system file to which you want to save the BITE log. If you do not specify a full path name, the system saves the file to the default BITE log directory, `$SINAP_HOME/Logs/bite`.

`filename`

Until the BITE log is saved to a system file, you can only examine it by means of BITE log-analysis commands. Once you save the log to a system file, you can examine it by using any of the system's standard display utilities (such as `vi`, `page`, or `cat`).

EXAMPLES

The following command displays the log file `test1.23sep`, which is located in the default directory for BITE log files (`$SINAP_HOME/Logs/bite`).

```
************************* BITE   LOG   ANALYSIS *************************
The Syntax of the MML Command is:

COMMAND:PARAMETER=[OPERATION]VALUE,...;
DISPLAY:file=/user/sinap/Logs/bite/test1.23sep
```

The following example shows a sample IPC data record. (See the `iblock.h` include file for a description of the fields in the `I_Block`, in which the IPC data is stored.)

```
********************        DISPLAY LOG RECORD FILE
********************
IPC Data:       Record= 0001
                Timestamp Index= 2
Time:   18:32:14:156 Tsid= 02 ( Appl=  BI, Proc=  LF )
Time:   18:32:14:156 Tsid= 01 ( Appl=  BI, Proc=  MI )

     Transaction: MSG= 0X00000003  REF= 5  MON= 2  SCEN= 0
     Originator:  NODE=  N1 MOD=  M1 APPL=  BI PROC=  MI INST= 1
     Destination: NODE=  N1 MOD=  M1 APPL=  NM PROC=  TH INST= 1
     Message:     MORE= 0 LEN= 201 RET CODE= 29
IPC data:
 e1 80 16 0b 4e 31 2c 4d 31 2c 42 49 2c 4d 49 16  - a...N1,M1,BI,MI.
 14 31 39 39 38 2d 31 31 2d 30 33 20 20 31 38 3a  - .1998-11-03  18:
 33 32 3a 31 34 16 00 02 01 06 16 32 53 54 41 52  - 32:14......2STAR
 54 2d 4d 4f 4e 3a 45 4e 54 3d 28 49 50 43 2c 4e  - T-MON:ENT=(IPC,N
 31 2c 4d 31 2c 42 49 2c 4d 49 29 2c 44 49 53 50  - 1,M1,BI,MI),DISP
 3d 59 2c 4c 4f 47 3d 69 70 63 2e 6c 6f 67 02 01  - =Y,LOG=ipc.log..
 1d 16 4c 2d 2d 2d 2d 74 69 6d 65 2d 2d 2d 2d 20  - ..L----time----
 2d 2d 2d 6f 72 69 67 2d 2d 2d 20 2d 2d 2d 64 65  - ---orig--- ---de
 73 74 2d 2d 2d 20 2d 2d 6d 73 67 5f 74 79 70 65  - st--- --msg_type
 2d 2d 20 73 69 7a 65 20 2d 2d 2d 2d 2d 2d 2d 2d  - -- size --------
 2d 2d 64 61 74 61 2d 2d 2d 2d 2d 2d 2d 2d 2d 16  - --data---------.
 00 16 00 16 00 00 00 f3 80 16 0c 20 20 42 49 2c  - .......s...  BI,
 20 20 4d 49 2c 4f 4b 00 00                       -   MI,OK..
********************        DISPLAY LOG RECORD FILE
********************
```

The following figure shows the format of an MSU data record. (For a description of the fields in the `M_Block`, which contains the MSU data, see the SINAP/SS7 `mblock.h` include file.)

```
********************        DISPLAY LOG RECORD FILE
********************
MSU data:       Record= 0001
                Timestamp Index= 3
Time:   17:54:49:156 Tsid= 04 ( Appl=    , Proc=     )
Time:   17:54:49:156 Tsid= 06 ( Appl=    , Proc=     )
Time:   17:54:49:000 Tsid= 07 ( Appl=    , Proc=     )

     Transaction: MSG= 0X00000105  REF= 0  MON= 1  SCEN= 0
     Originator:  NODE=  N1 MOD=  M1 APPL=  BI PROC=_LNK INST= 0
     Destination: NODE=  N1 MOD=  M1 APPL=  BI PROC=   0 INST= 0
     Message:     MORE= 0 LEN= 300 RET CODE= 0
M_BLOCK header:
        BITE_Control: CMD= 0  QUAL= 0  LINK= 0  PID1= 0
                      PID2= 0 RW= 0   MON= 0
        CASL_Control: LOST CNT= 00 PID= 0  LINK= 0x0000  SENDER= 0
                      IBLK= 0 RW= R   MON= 1   SSN_SIO= 0
        TCAP_Control: MSG= 0 TRANS= 00 ABORT TYPE= 0 ABORT CAUSE= 0
        SCCP_Control: CTRL= 0 SRC= 0 DEST= 0 SLS5= 0x00 ERROR= 0 PRIO= 0
SEQ= 0
        MTP_Control:  MSGID= 0x11 SEQID= 00 MSGTYPE= 5 SENDID= 170 MSG
SIZE= 124                      MTP user data: (HEX)  00 00 12 22 20 00
f4 a0
SS#7 data:
        L2:  BIB= 179 FIB= 181 LI= 63
       SIO= 0x03 (NI= 00, SI= 03)    SCCP
        DPC= 01-119-03(3003) OPC= 01-085-02(2730) SLS= 06
Message Block: (HEX)     Length= 116
--------------
 09 80 03 07 0b 04 43 bb 0b fd 04 43 aa 0a fe 64  - ......C;.}.C*.~d
 62 62 48 04 00 1e 00 ab 6c 5a a1 2b 02 01 66 06  - bbH....+lZ!+..f.
 02 83 01 f2 22 aa 0b 84 09 01 00 21 0a 08 60 06  - ...r"*.....!..`.
 99 11 84 07 02 00 21 06 02 21 42 84 06 07 00 01  - ......!..!B.....
 03 22 04 df 45 01 17 a1 2b 02 01 67 06 02 83 01  - ."._E..!+..g....
 f2 22 aa 0b 84 09 01 00 21 0a 08 60 06 99 11 84  - r"*.....!..`....
 07 02 00 21 06 02 21 42 84 06 07 00 01 03 22 04  - ...!..!B......".
 df 45 01 17                                      - _E..
```

NOTES

Each record contains up to eight time stamp fields. The log-analysis program displays the last logged time first and the first logged time last, and it extracts the records according to the last logged time.

FIND

SYNOPSIS

FIND:FILE=*logfile*,OFILE=*file*,*key*=*key_value*;

DESCRIPTION

This FIND command extracts the records from a log file that satisfy all criteria specified in the command's arguments. The command has the following arguments.

logfile
: Specifies the name of the log file to open by searching the Logs/bite directory for the appropriate log file.

file
: Specifies the name of the file to which extracted records are saved.

key
: Specifies a key word from the fields in the log file record. For permissible values, see Table 4-3 earlier in this chapter.

key_value
: Specifies the value to search for in the MSU; for example, a beginning and ending time range (BT and ET), destination point code (DPC), and signaling information octet (SIO). The value of *key_value* is dependent on the value of *key*.

You can specify more than one *key* and *key_value*.

EXAMPLES

The following example extracts all the records in the LOGSS710 log file that begin at 12:20, end at 2:30, have a DPC of 1-1-8, an OPC of 1-1-8, and contain an SIO of 3. The records that meet these criteria are written to the file called TSTSUM.

```
FIND:FILE=LOGSS710,OFILE=TSTSUM,BT=12:20:00,ET=2:30:00,DPC=1-1-8,OPC=1-1-8,SIO=3;
```

SELECT

SYNOPSIS

SELECT:FILE=*logfile*,OFILE=*file*,*key*=*key_value*;

DESCRIPTION

The SELECT command extracts the records from a log file that satisfy any of the criteria specified in the command's arguments. The command has the following arguments.

logfile
: Specifies the name of the log file to open.

file
: Specifies the name of the file to which extracted records are saved.

key
: Specifies a key word from the fields in the log file record. Valid values are listed in Table 4-3 earlier in this chapter.

key_value
: Specifies the value to search for in the MSU; for example, a beginning and ending time range (BT and ET), destination point code (DPC), and signaling information octet (SIO). The value of *key_value* is dependent on the value of *key*.

You can specify more than one *key* and *key_value*.

EXAMPLES

The following example extracts all records from the LOGSS710 log file that have a DPC of 1-1-8, or an OPC of 1-1-8. The records that meet either of these criteria are written to the tstsum file.

```
SELECT: FILE=LOGSS710,OFILE=TSTSUM,DPC=1-1-8,OPC=1-1-8
```

SUMMARY

SYNOPSIS

SUMMARY:FILE=*logfile*,OFILE=*file*,*key*=*key_value*;

DESCRIPTION

The SUMMARY command counts the records in the specified log file. A record is counted if it contains the values specified in the command's arguments. The command has the following arguments.

logfile
: Specifies the name of the log file to open.

file
: Specifies the name of the file to which extracted records are saved.

key
: Specifies a key word from the fields in the log file record. For a list of valid values, see Table 4-3 earlier in this chapter.

key_value
: Specifies the value to search for in the MSU; for example, a beginning and ending time range (BT and ET), destination point code (DPC), and signaling information octet (SIO). The value of *key_value* is dependent on the value of *key*.

You can specify more than one *key* and *key_value*.

EXAMPLES

The following example counts all records from the record log called LOGSS710, using the key values MSG and CA_LNK, and writes these records to a file called TSTSUM.

```
SUMMARY:FILE=LOGSS710,OFILE=TSTSUM,MSG=X105,CA_LNK=32;
******************** LOG FILE SUMMARY File:TSTSUM ********************
FILE/home/sinap/Logs/bite/LOGSS710
Total records: 2
MSG:           2
CA_LNK:        1

EOF
```

QUIT

SYNOPSIS

`QUIT`

DESCRIPTION

This command terminates the Log Analysis program, returning you to the SINAP/SS7 login window from which you invoked the program.

NOTES

You can specify the `QUIT` command as:

- `QUIT:;`
- `QUIT:`
- or `QUIT`

BITE Commands Reference

This section describes each of the following BITE MML commands. The commands are presented in alphabetical order.

- `DISPLAY-SCEN` displays information about all active scenarios.
- sends a debug command to a process.
- initiates a monitor process, in which the BITE collects information about the specified entities.
- initiates a scenario using the BITE scenario execution program.
- stops the specified BITE monitor process.
- stops the specified scenario.

DISPLAY-SCEN

SYNOPSIS

```
DISPLAY-SCEN;
```

DESCRIPTION

This command displays information about active scenarios. (You initiate a scenario by means of the BITE scenario-execution program.) You can use the `DISPLAY-SCEN` command to obtain the scenario ID assigned to the scenario; you will need this ID when you issue the MML command `STOP-SCEN` to terminate the scenario. The `DISPLAY-SCEN` command displays a line of information about each active scenario; the scenario's ID is the first thing listed. (For more information about scenarios and scenario execution, see "Scenario Execution" earlier in this chapter.)

The `DISPLAY-SCEN` command has no arguments.

EXAMPLES

The following is sample output of `DISPLAY-SCEN`. The scenario shown in the following example has a scenario ID of `1`.

```
DISPLAY-SCEN;
current active scenarios:

1 ENT=(,,CLIENT_SCEN,PROCESS_1,1),FILE=FILENAME;
```

NOTES

The alternative and man page format for `DISPLAY-SCEN` is `DISPL-SCEN`. This command is accessible from the BITE Commands menu.

START-DBG

SYNOPSIS

START-DBG:ENT=(*entity*),MSG=*message*;

DESCRIPTION

This command sends a debug message to the specified process. The command has the following arguments.

entity

Specifies the node, module, application, process names, and instance number of the process to receive debug messages. The format of *entity* is

ID0,ID1,ID2,ID3,ID4

where *ID0* is the node name, *ID1* is the module name, *ID2* is the application name, *ID3* is the process name, and *ID4* is the instance number. The commas are part of the format of *entity*.

message

Specifies the content of the message being sent to the destination process. (Message content depends on the destination process.) The destination process must intercept the message. For example, you can turn on a debug mask so that your process sends related debug messages to the BITE, which can then display them at the terminal or write them to a log.

You must specify the *message* argument as the last argument in the command line. Because the semicolon (;) is the terminating character for the command, you cannot use it in your message.

EXAMPLES

The following example shows the START-DBG command for an application with a registered name of APPL, a process name of PROC, and a message that is a debug mask. Both the application and process are on the same node and module.

```
START-DBG:ENT=(N1,M1,APPL,PROC,1),MSG=debug_mask 0x00000ff0;
```

NOTES

The alternative and man page format for START-DBG is STA-DBG. This command is accessible from the BITE Commands menu.

START-MON

SYNOPSIS

START-MON:ENT= (*entity*)[(*&entity*)][,DISP=Y/N] ,LOG=*filename* [,CONT=Y/N];

DESCRIPTION

This command initiates monitoring for specified entities from any SINAP/SS7 process. If no errors are detected, the command returns a monitor ID. The command has the following arguments.

entity

Specifies the type of message (IPC, SS7, or link) and process or link identities to be monitored. The format of *entity* is

TYPE,*ID0*,*ID1*,*ID2*,*ID3*,*ID4*,[R/W]

where *TYPE* can have the value IPC, SS7, or LNK.

If *TYPE* is IPC or SS7, *ID0* represents the node name, *ID1* represents the module name, *ID2* represents the application name, *ID3* represents the process name, and *ID4* represents the instance number. However, *ID2* and *ID3* are required, and *ID0*, *ID*1, and *ID4* are optional. The default node name is the name of the current node. The default module name is the current local module. If you omit the *ID4* parameter, the SINAP/SS7 system monitors all current instances. Any instance created after the command is executed is not monitored.

If *TYPE* is LNK, *ID0* represents the link number. *ID1, ID2*, *ID3*, and *ID4* are not present.

To monitor different paths using the same log file, you can specify more than one entity by separating each with an ampersand (&).

The R and W arguments are optional. Specifying R indicates a monitoring read; specifying W indicates a write operation. If you omit either option, the SINAP/SS7 system assumes that both read and write operations are desired. You can specify up to eight entities.

DISP

Determines whether the monitored events should be displayed on the operator terminal. Fields displayed include the application name and process name of the message originator, the application name and process name of the message's destination, the message type, the last time stamp, data size, and first eight bytes of data. You must enter either Y or N.

filename

Specifies the log file name in which the monitored events are logged. The SINAP node writes all log files to the directory Logs/bite. If the storage capacity is exceeded, the node discontinues the log activity and sends an alarm to Trouble Management. The default

size for this log is 200K bytes.

In addition to the limitation that the `Logs/bite` partition imposes for all BITE log files, there is a default size limit for each log file. If a log file reaches the limit, the SINAP/SS7 system automatically closes it and sends an alarm to Trouble Management.

`CONT`

Specifies whether continuous logging should be enabled. Normally, the BITE stops monitor output to a log file when the file size surpasses 200K. The `CONT` argument allows you to specify that data beyond 200K should be saved. When the BITE log file reaches its limit and `CONT` is specified, the file is renamed to a backup file, and a new logging file is opened. The backup file is of the format *`filename`*`.bak`. The system deletes any previous backup file with the same name.

Logging continues until you stop it with the `STOP-MON` command.

To use this argument, you must specify a value for *`filename`*. The default value for this argument is `N`.

EXAMPLES

The following is sample output of `START-MON`.

```
 START-MON:ENT=(IPC,,,NM,CL),DISP=Y;

----time----     --orig---  --dest---  --msg_type---  --size--- ------data----

NM,CL,OK
0007:51:34:105     NM,CL1     NM,CM1       RUOK   000R
0107:51:34:105     NM,CM1     NM,CL1       IMOK   000W
0207:52:34:103     NM,CL1     NM,CM1       RUOK   000R
0307:52:34:103     NM,CM1     NM,CL1       IMOK   000W
```

The values in the `time` field represent the last time stamp during monitoring.

The values in the `orig` field represent the originator application name and process name. In this example, Node Management is the application name, and the Node Management Client Management and Command Management processes represent the process names.

The values in the `dest` field represent the destination application name and process name. In this example, the Node Management is the application name, and the Node Management Client Management and Command Management processes represent the process names.

The `msg_type` field represents the type of message being sent, for example, `RUOK` and `IMOK`. In the example, because health check operations are enabled, health check messages appear in the field.

The `size` field represents the data size for the `M_Block` or `I_Block`. Because no messages are being transmitted, this field is blank.

The `data` field represents the first eight bytes of the message; in this case, they do not exist.

NOTES

The alternative and man page format for `START-MON` is `STA-MON`. This command is accessible from the BITE Commands menu.

START-SCEN

SYNOPSIS

`START-SCEN:ENT=(`*`entity`*`),FILE=`*`filename debug_mask test_MSU`*`;`

DESCRIPTION

This command starts a scenario and returns the scenario ID it assigned to the scenario. For more information about scenarios and scenario execution, see "Scenario Execution" earlier in this chapter.

The command has these arguments.

`entity`

Specifies the node, module, application, process names, and instance number of the process being tested. The format of *`entity`* is

`ID0,ID1,ID2,ID3,ID4`

where *`ID0`* is the node name, *`ID1`* is the module name, *`ID2`* is the application name, *`ID3`* is the process name, and *`ID4`* is the instance number. The commas are included in the format of *`entity`*.

For remote testing, or for testing routing, do not specify an entity. The scenario execution program must register with the SINAP/SS7 system as a signaling information octet (SIO) application or an SSN application. For local test, use your own PC for the DPC. For remote testing, the DPC in the MSU should indicate the destination PC.

`filename`

Specifies the name of the scenario-execution file you want to run for this scenario execution.

`debug_mask`

Specifies a debug mask to be used for the scenario execution (for example, `0xffff`). For a list of valid values, see the `cadbg.h` include file.

`test_MSU`

Specifies the name of the Database Builder program file that contains the test MSU you want to use for the scenario execution. Because the semicolon (`;`) is used to terminate the MML `START-SCEN` command, you cannot use it in your test MSU. For information about creating this file, see "Using the Database Builder to Create Test MSUs" earlier in this chapter.

NOTES

The alternative and man page format for `START-SCEN` is `STA-SCEN`. This command is accessible from the BITE Commands menu.

STOP-MON

SYNOPSIS

`STOP-MON:ENT=`*`monitor_id`*`;`

DESCRIPTION

This command stops monitoring for specified entities.

The *`monitor_id`* argument specifies the entity to be stopped. Use the value the `START_MON` command returns for this argument, or use the `DISPLAY_MON` command to see all active monitor IDs.

NOTES

This command is accessible from the BITE Commands menu.

STOP-SCEN

SYNOPSIS

`STOP-SCEN:ENT=`*`scenario_id`*`;`

DESCRIPTION

This command stops a specified scenario execution.

The *`scenario_id`* argument specifies the ID of the scenario you want to stop. For this argument, specify the value returned by the `START-SCEN` command, or use the `DISPLAY_MON` command to see the IDs assigned to all active scenarios.

NOTES

This command is accessible from the BITE Commands menu.

Measurement Collection Commands

Although not a part of the BITE subsystem, the following MML commands are useful for collecting measurements related to the number and types of messages that the SINAP/SS7 system sends and receives. When you issue a measurement reporting command, you use the start and stop arguments in the commands to specify the time period you want the report to cover. The commands generate a report presenting the statistics gathered for that time period. You can generate a report for a particular time period, such as a day, a week, a month, or longer.

You must set the `SINAP_ALT_MEASUREMENT_INTERVAL` environment variable **before** starting the SINAP/SS7 system to define the measurement interval you want to use according to Table 4-4.

Table 4-4. Setting Measurement Intervals

Setting	Measurement Interval
`0` or `30`	30 Minutes (default)
`1` or `15`	15 Minutes
`2` or `5`	5 Minutes

For example, if you set it to 1 or 15, a 15-minute measurement interval is used. A 15-minute interval allows you to produce reports for any time period between 15 minutes and one year. If you set it to 0 or 30, the SINAP/SS7 system uses a 30-minute interval and produces reports for 30-minute time periods. The system default is 30 minutes if no time is defined for this environment variable.

You can save a measurement report to a file by specifying the file name and location using the `Print to Filename` argument. You can then print and view particular reports when needed. See the section in Chapter 2 of the *SINAP/SS7 User's Guide* (R8051) on enabling and disabling the printing functions for information on activating the print options in the SINAP/SS7 system.

For each command, you can define the time period the report covers, generate a report that presents the statistics or measurements gathered, save the report to a file, and print a copy of the report.

You specify the time and date information using these guidelines:

- For the start and end date, use the format `[CC]YY-MM-DD` where `[CC]YY` is the century and the year, for example, 1998.(Optionally, you can enter only a two digit year, for example, 98.) `MM` is the month, for example, 01 for January. `DD` is the date, for example, 15. Include hyphens between the values, as in 1998-01-15, or optionally, 98-01-15.

  Valid values for the year arguments include:

- CC = 19 or 20
- YY = 80 through 99 for century 19, or 0 through 38 for century 20
- MM = 01 through 12
- DD = 01 through 31

To generate a report for today's measurements, you can enter the value, TODAY, as the date argument instead of using the format [CC]YY-MM-DD. If you do not specify an end date, the command uses the same date as the start date.

- For the start and end time, use the format HH:MM where HH is the hour and MM is the minutes. Include a colon between the values. If no end time is specified, the command generates a report for a 30-minute period, beginning at the specified start time. For example, if you specify 12:00 as the start time and do not specify an end time, the command generates a report for the 30-minute time period between 12:00 and 12:30.

  Valid values for the start and end times are:

  - HH = 1 through 24
  - MM = Either 00 or 30

Report Measurement Considerations

When you issue commands to generate a measurement report, consider the following:

- Measurements are only generated while the SINAP node is active. If you issue a measurement-reporting command for a period of time during which the SINAP node was inactive, the command generates an empty measurement report.

  If the SINAP node was active for **any** amount of time during the specified time period, the measurement report will contain data, but it will not be obvious from the data how much of the time the SINAP node was active. If the measurement report contains MTP statistics, you can determine the amount of time that the SINAP node was active from the L2 Serv field of the MTP measurements section. The L2 Serv field shows the amount of time (in seconds) that the SINAP node was active during the time period covered by the measurement report. For example, suppose you generate an MTP measurement report for a 24-hour period of time during which the SINAP node was active for only 12 hours. Although the report appears to contain data for the specified 24-hour time period, the L2 Serv field indicates that the SINAP node was only active for 12 of the 24 hours.

- A blank measurement report indicates there is no measurement data for the specified time interval. This indicates one of the following conditions: the SINAP node was not running during the specified time period, the SINAP node was running but the measurement-collection process was turned off, or the log file containing those measurements was deleted from the $SINAP_HOME/Logs/system directory.
- If you issue one of the measurement-reporting commands and specify an invalid period of time, the command returns an error message.

Saving the Report to a File and Printing It

You can save the measurement report to a file and print it, using the following instructions:

1. Display the measurement report you want to save using the appropriate command.
2. Press CTRL-P to invoke the Print Options menu.
3. Select the Print to Filename option.
4. Specify a filename and press RETURN. The file saves by default into the `/home/sinap/sysopr` directory. If you want to save it to another location, include the file path in the file name specification. For example, specifying the file path, `FILE=/home/sinap/mtp-1230`, writes the measurement file, `mtp-1230`, to the home directory of the SINAP/SS7 login account, `/home/sinap`.

You can report measurements for each MTP, SCCP, and TCAP subsystem or for all systems.

The measurements commands are described in the following sections in alphabetical order.

- `DUMP-TABLE` dumps (saves) the contents of the MTP routing and management tables to the static table file.
- `REPORT-MALL` reports the results of all measurements related to the MTP, SCCP, and TCAP subsystems.
- `REPORT-MMTP` creates a report on MTP-related information.
- `REPORT-MSCCP` creates a report on SCCP-related measurements.
- `RETRIEVE-NOM` retrieves the oldest 5, 15, or 30-minute node network management report.
- `RETRIEVE-SMR` retrieves the most recently compiled 5-minute node network management report.
- `REPORT-MTCAP` creates a report on TCAP-related measurements.
- `START-MEASURE` starts a measurement-collection process.
- `START-MWRITE` starts a measurement-write process in which measurements are written to a measurement log.
- `STOP-MEASURE` stops a measurement-collection process.
- `STOP-MWRITE` terminates a measurement-write process.

DUMP-TABLE

SYNOPSIS

`DUMP-TABLE;`

DESCRIPTION

The Dump Table (`DUMP-TABLE`) command dumps (saves) the contents of the MTP routing and management tables to the static table file, (`$SINAP_HOME/cm_display/static_table`), in binary format. The Stratus Customer Assistance Center (CAC) can use this file to aid in problem analysis. This command has no arguments.

EXAMPLES

The following is sample output from the `DUMP-TABLE` command.

```
M  DUMP TABLE                                    ok
   table dumped in file:home/sinap/cm_display/static-table
```

NOTES

The alternative and man page format for this command is `dump-table`.

REPORT-MALL

SYNOPSIS

REPORT-MALL:DATE=*date*,TIME=*time*[,PRINT=*print*],[FILE=*file*];

DESCRIPTION

This command reports the results of all measurements related to the MTP, SCCP, and TCAP subsystems. Specify the date and time as described previously. For the print argument, specify YES for printing to the default printer, NO for no printing, or the printer ID for printing to a printer.

EXAMPLES

The following sample output of REPORT-MALL shows the measurements for all subsystems for the 30-minute time period starting at 8:30 AM on June 24:

```
M  REPORT-MALL:DATE=TODAY,TIME=08:30;
   command completed

Report MTP 30 minute measurements:
MTP 30 minute measurements: 06/24, 08:30

Link Name  L3 Unava   L3 Cong     L3 TX   L3 RX   L2 Serv  L2 SU Err
LNKA00            0         0         0       0         0          0
LNKA02            0         0         0       0         0          0
LNKA03            0         0         0       0         0          0
LNKA04            0         0         0       0         0          0

SP     MSU Discarded

Report SCCP Measurements:

SCCP Measurements: 06/24, 08:30

PC Not Available = 0
Network Configuration = 0
SSN Not Available = 0
Unequipped User = 0
Syntax Error = 0
Unknown Reason = 0

SSN    Message Sent     Message Received
254              0                    0

Report TCAP Measurements:

TCAP Measurements: 06/24, 08:30

SSN   Comp Sent    Comp Rcvd    Local Reject    Return Error
254           0            0               0               0
```

NOTES

The alternative and man page format for the REPORT-MALL command is `rept-mall`.

REPORT-MMTP

SYNOPSIS

REPORT-MMTP:DATE=*date*,TIME=*time*[,PRINT=*print*],[FILE=*file*];

DESCRIPTION

This command creates a report on MTP-related information. Specify the date and time as described previously. For the print argument, specify YES for printing to the default printer, NO for no printing, or the printer ID for printing to a printer.

EXAMPLES

The following sample report shows measurement data of REPORT-MMTP for a 30-minute time period beginning at 8:30 AM on June 24:

```
REPORT-MMTP:DATE=TODAY,TIME=08:30;
command completed

MTP 30 minute measurements: 06/24, 08:30

Link Name  L3 Unava   L3 Cong    L3 TX   L3 RX  L2 Serv  L2 SU Err
LNKA00            0         0        0       0        0          0
LNKA02            0         0        0       0        0          0
LNKA03            0         0        0       0        0          0
LNKA04            0         0        0       0        0          0

SP     MSU Discarded
```

NOTES

The alternative and man page format for REPORT-MMTP is rept-mmtp.

REPORT-MSCCP

SYNOPSIS

REPORT-MSCCP:DATE=*date*,TIME=*time*[,PRINT=*print*],[FILE=*file*];

DESCRIPTION

This command reports SCCP-related measurements. Specify the date and time as described previously. For the print argument, specify YES for printing to the default printer, NO for no printing, or the printer ID for printing to a printer.

EXAMPLES

The following sample output of REPORT-MSCCP shows an SCCP report for a 30-minute report interval beginning at 8:30 AM on June 24. User entries are in bold type.

```
REPORT-MSCCP:DATE=TODAY,TIME=08:30;
command completed
SCCP Measurements: 06/24, 08:30

PC Not Available = 0
Network Configuration = 0
SSN Not Available = 0
Unequipped User = 0
Syntax Error = 0
Unknown Reason = 0

SSN    Message Sent     Message Received
254              0                    0
```

NOTES

The alternative and man page format for REPORT-MSCCP is rept-msccp.

REPORT-MTCAP

SYNOPSIS

REPORT-MTCAP:DATE=*date*,TIME=*time*[,PRINT=*print*],[FILE=*file*];

DESCRIPTION

This command reports TCAP-related measurements. Specify the date and time as described previously. For the print argument, specify YES for printing to the default printer, NO for no printing, or the printer ID for printing to a printer.

EXAMPLES

The following sample output of REPORT-MTCAP shows a TCAP report for a 30-minute interval beginning at 8:30 AM on June 24.

```
REPORT-MTCAP:DATE=TODAY,TIME=08:30;
command completed

Report TCAP Measurements:

TCAP Measurements: 06/24, 8:30

SSN   Comp Sent    Comp Rcvd    Local Reject    Return Error
254           0            0               0               0
```

NOTES

The alternative and man page format for REPORT-MTCAP is rept-mtcap.

RETRIEVE-NOM

SYNOPSIS

```
RETRIEVE-NOM;
```

DESCRIPTION

The Retrieve Oldest 5, 15, or 30-Min Measurement (`RETRIEVE-NOM`) command retrieves the oldest 5, 15, or 30-minute node network management measurement report. After it displays, the report is deleted. The output for this command is similar to the output for `REPORT-MALL`. (See the previous section on reporting measurements for MTP, SCCP, and TCAP.)

The SINAP/SS7 system collects statistical information on the number of times and the length of time that destination point codes (DPCs) are inaccessible during a 5, 15, or 30-minute period. The SINAP/SS7 system keeps track of the route set used to access individual DPCs by applying a *timestamp* (time received) to each route set. The SINAP/SS7 system also maintains two structures, `rcMeasData` and `routeSetMeasurements`, for each route set. The structures contain inaccessibility information.

> N O T E
>
> When the SINAP/SS7 system first starts, the system marks each route set as inaccessible and sets the timestamp of each route set to the time that the SINAP/SS7 system was started. When the route set first becomes accessible, the SINAP/SS7 system uses this timestamp to measure the length of time that route set was initially inaccessible.

During each 5, 15, or 30-minute measurement period, the SINAP/SS7 `l3rc` process collects inaccessibility data for each route set and stores this data in static memory in the route set's `rcMeasData` structure. At the end of the measurement period, the contents of `rcMeasData` are written to the route set's `routeSetMeasurements` structure, which is stored in shared memory. The SINAP/SS7 system re-initializes the `rcMeasData` structure's fields to zero and begins collecting inaccessibility data for the next 5, 15, or 30-minute measurement period.

> N O T E
>
> To display the measurements for a particular destination point code (DPC), issue the `REPORT-MMTP` command *or* the `REPORT-MALL` command and specify the current date and time. The SINAP/SS7 system displays the DPC's inaccessibility measurements for the preceding 5, 15, or 30-minutes. The first column, `Destination`, indicates the DPC whose measurements are displayed. The second and third columns indicate the number of times and the total amount of

time, respectively, that the DPC was inaccessible during the preceding 5, 15, or 30-minutes.

There are no arguments for this command.

NOTES

The alternative and man page format for `RETRIEVE-NOM` is `rtrv-nom`.

EXAMPLES

The following sample screen shows the 30-minute network management measurement report for the RETRIEVE-NOM command.

```
M  RETRIEVE-NOM;
   command completed

Report MTP 30 minute measurements:
MTP 30 minute measurements: 12/14, 00:00 - 12/14, 00:30

Link Name  L3 Unava   L3 Cong     L3 TX   L3 RX   L2 Serv  L2 SU Err
LNK0            1800          0        0        0          0          0
  Number of MSUs:                       0        0
LNK1            0             0        0        0          0          0
  Number of MSUs:                       0        0
LNK2            0             0        0        0          0          0
  Number of MSUs:                       0        0
LNK3            0             0        0        0          0          0
  Number of MSUs:                       0        0
LNK4            0             0        0        0          0          0
  Number of MSUs:                       0        0
LNK5            0             0        0        0          0          0
  Number of MSUs:                       0        0
LNK6            0             0        0        0          0          0
  Number of MSUs:                       0        0
LNK7            0             0        0        0          0          0

SP     MSU Discarded
    Destination        Occurrences          Duration
            3003                0              1800
Report SCCP Measurements:

SCCP Measurements: 12/14, 00:00 - 12/14, 00:30

PC Not Available = 0
Network Configuration = 0
SSN Not Available = 0
Unequipped User = 0
Syntax Error = 0
Unknown Reason = 0

Report TCAP Measurements:

TCAP Measurements: 12/13, 23:30 - 12/14, 00:00

SSN   Comp Sent    Comp Rcvd    Local Reject    Return Error
254           0            0               0               0
```

RETRIEVE-SMR

SYNOPSIS

`RETRIEVE-SMR;`

DESCRIPTION

The Retrieve Latest 5, 15, or 30-Minute Measurement (`RETRIEVE-SMR`) command retrieves the most recently completed node network management measurement report, including the beginning and ending times of the period. After the report displays, it is deleted. You cannot save or print the output. If there are any measurements older than the ones being retrieved, they are also deleted.

EXAMPLES

The following sample output shows a 5-minute network management report.

```
M     RETRIEVE-SMR;
      command completed

Report MTP 5-minute measurements:
MTP 5 minute measurements: 11/17. 19:12

      Adjacent SP      Unavailable
              3003            299
                 0              0
                 0              0
                 0              0
                 0              0
                 0              0
```

NOTES

The alternative and man page format for `RETRIEVE-SMR` is `rtrv-smr`.

START-MEASURE

SYNOPSIS

START-MEASURE:MEASURE=*measure*;

DESCRIPTION

This command starts on-demand measurements and reports the data that is collected by the measurements collection process. The *measure* argument specifies the on- demand measurements to be initiated. Possible values for *measure* are as follows:

OCTXMTsignaling Information Field (SIF) and Service Information Octet (SIO) transmitted.

OCTRCVSIF and SIO octets received.

EXAMPLES

The following is sample output from the Measurement Collection Process.

```
Description                                Level    Units      Duration    Activation
Duration of Link in the In-Service State    MTP 2   seconds/SL  30 minute   Permanent
Number of Signal Units in Error             MTP 2   events/SL   30 minute   Permanent
Duration of SL Unavailability               MTP 3   seconds/SL  30 minute   Permanent
Number of SIF and SIO Octets Transmitted    MTP 3   octets/SL   30 minute   On Demand
Number of SIF and SIO Octets Received       MTP 3   octets/SL   30 minute   On Demand
MSUs discarded Due to SL Congestion         MTP 3   MSUs/SL     30 minute   Permanent
Duration of Adjacent SP Inaccessible        MTP 3   seconds/SP   5 minute   Permanent
MSUs Discarded Due to a Routing Data FailureMTP 3   MSUs/SP      5 minute   Permanent
Routing Failure-Point Code Not Available    SCCP    messages    30 minute   Permanent
Routing Failure-Network Congestion          SCCP    messages    30 minute   Permanent
Routing Failure-Subsystem Unavailable       SCCP    messages    30 minute   Permanent
Routing Failure-Unequipped User             SCCP    messages    30 minute   Permanent
Syntax Error Detected                       SCCP    messages    30 minute   Permanent
Routing Failure-Reason Unknown              SCCP    messages    30 minute   Permanent
Total Messages Sent                         SCCP    msgs/appl   30 minute   Permanent
Total Messages Received                     SCCP    msgs/appl   30 minute   Permanent
Total TCAP Components Sent                  TCAP    comp/appl   30 minute   Permanent
Total TCAP Components Received              TCAP    comp/appl   30 minute   Permanent
TCAP Local Rejects                          TCAP    rej/appl    30 minute   Permanent
TCAP Return Errors                          TCAP    err/appl    30 minute   Permanent
```

Key:
SL = signaling Link SP = signaling Point
SIF = signaling Information Field SIO = Service Information Octet
For further information, see ANSI Recommendation T1.111.6

NOTES

The alternative and man page format for START-MEASURE is sta-measure.

START-MWRITE

SYNOPSIS

```
START-MWRITE;
```

DESCRIPTION

This command initiates writing of measurements to the measurement logs in the `$SINAP_HOME/Logs/system` directory. There are no arguments for this command.

NOTES

The alternative and man page format for `START-MWRITE` is `sta-mwrite`.

STOP-MEASURE

SYNOPSIS

`STOP-MEASURE:MEASURE=`*`measure`*`;`

DESCRIPTION

This command stops on-demand measurements.

The *`measure`* argument specifies the on-demand measurements to be stopped. Valid values are as follows:

`OCTXMT` SIF and SIO octets transmitted

`OCTRCV` SIF and SIO octets received

NOTES

The alternative and man page format for this command is `stop-measure`.

STOP-MWRITE

SYNOPSIS

```
STOP-MWRITE;
```

DESCRIPTION

This command stops measurement writing to the measurement logs in `$SINAP_HOME/Logs/system`. The command has no arguments.

NOTES

The alternative and man page format is `stop-mwrite`.

Chapter 5
Sample Applications

This chapter presents samples of different types of SINAP/SS7 applications. All the network variants support sample TCAP applications. The CCITT and China network variants also support sample SCCP and MTP applications. CCITT and ANSI variants support sample ISUP applications. For information on sample ISUP applications, see the *SINAP/SS7 ISDN User Part (ISUP) Guide* (R8053).

The following sections cover all the sample application programs for the different network variants:

- "Compiling the Sample Applications"
- "Sample TCAP Application"
- "Sample SCCP Applications"
- "Sample MTP Applications"

By default, these sample applications are located in the following directory:

```
$SINAP_HOME/Samples/<network variant>
```

For example, the sample applications for the CCITT network variant are located in the directory:

```
$SINAP_HOME/Samples/ccitt
```

The sample program directory contains *executable* and *non-executable* programs. The sample program names with a `.c` suffix (for example, `tcsend.c`) are non-executable. The sample application program names without the `.c` suffix (for example, `tcsend`) are the executable programs. To execute the sample programs, move to the directory in which the programs are located and enter the name of the executable program (**without** the `.c` suffix). For example, to execute the TCAP sample program `tcsend`, enter the command `tcsend` from the directory in which the program is located, for example:

```
$SINAP_HOME/Samples/ccitt/
```

To change any sample program provided with the SINAP/SS7 software, make a copy of the program, then make the changes to the copy. After making modifications, compile the modified program to create a new executable file. See the following section, "Compiling the Sample Applications," for instructions.

Compiling the Sample Applications

The sample applications, provided in the SINAP Installation package, are compiled differently depending on your operating system.

Solaris Operating Systems

The 64-bit release version of the SINAP/SS7 software for Solaris/SPARC is designed to run on a Sun Netra 20/T4 or SunFire V480 series platform with Solaris 8 installed. The Solaris 8 kernel runs in 64-bit mode. The SINAP processes run in 32-bit mode, however, SINAP user applications are supported in either 32-bit or 64-bit mode. This is made possible by providing SINAP CASL and ISSL libraries in both 32-bit and 64-bit modes.

NOTE

The SINAP SS7-over-IP and `LSSUTIL` libraries are only supported in 32-bit mode.

At `$SINAP_HOME/Samples/ansi` and `$SINAP_HOME/Samples/ccitt` directories, the `Makefile` (for making 32-bit sample test programs) and `Makefile.64bit` (for the 64-bit mode) provide examples of CC_XOPTS, `CFLAGS` and `LDFLAGS` settings for making 32-bit and 64-bit mode executables respectively. The default, 32-bit mode, sample test programs, are provided in the SINAP release, however, they can be replaced with 64-bit mode executable versions, by executing "`make -f Makefile.64bit`" after "`make clean`" from the `$SINAP_HOME/Samples/ansi` or `$SINAP_HOME/Samples/ccitt` directory. Similarly, 32-bit mode sample test programs can be restored by executing the "`make clean`" then "`make`" or "`make -f Makefile`" scripts.

HP-UX Operating Systems

The SINAP/SS7 software on the 64-bit mode HP-UX operating system has all 32-bit SINAP libraries installed under `$SINAP_MASTER/Library` and 64-bit SINAP libraries installed under `$SINAP_MASTER/Library/64bit`. Note that SINAP/SS7 on the 64-bit mode HP-UX operating system has only 64-bit SINAP libraries and all SINAP processes are built for 64-bit mode. SINAP releases prior to SINAP 11.0 on 32-bit HP-UX operating systems have only 32-bit SINAP libraries, which are under `$SINAP_MASTER/Library`, and 32-bit SINAP processes.

Stratus ft Linux Systems

On Stratus ft Linux systems, all libraries and sample programs are 32-bit.

Sample Applications

Two makefiles, Makefile and Makefile.64bit, are provided under the SINAP samples directories of ANSI and CCITT variants (i.e. `$SINAP_HOME/Samples/[ansi, ccitt]`,

to make 32-bit and 64-bit sample test programs similar to the ones described above in the 'Solaris Operating Systems' section.

The `tcsend.c` and `tcrecv.c` sample programs contain calls to CASL functions. The sample programs also rely on specific SINAP/SS7 header files, libraries, and external routines. Therefore, when compiling `tcsend.c` or `tcrecv.c`, perform the following steps. (These steps are based on the assumption that the programs are located in the directory `$SINAP_HOME/Samples/<network variant>`.)

- Log in as the user, `sinap`.
- Issue the compile command (`cc`) from the directory in which the sample programs are located (by default, `$SINAP_HOME/Samples/<network variant>`).
- Specify the directory `$SINAP_HOME/Include` in the compile command line. `$SINAP_HOME/Include` contains the header files used by the sample programs.
- Specify the directory `$SINAP_HOME/Library` in the compile command line. `$SINAP_HOME/Library` contains the CASL library, which contains the CASL calls made by the sample programs.
- Specify the object file `tcap_2.o` in the compile command line. This object file defines the external routine `print_comp`, which `tcsend.c` and `tcrecv.c` use to handle their output. (The `tcap_2.c` program is also located by default in the directory `$SINAP_HOME/Samples`.)

As an alternative, this can be done by utilizing `$SINAP_HOME/Samples/<network variant>/Makefile` to compile a sample program. For example, issue "`make tcsend`" or "`make tcrecv`" or recompile everything under that directory by issuing the "`make`" after "`make clean`" command.

The following examples illustrate the commands you issue to compile the sample program `tcsend.c` on different platforms.

1. For HP-UX 32-bit mode:

```
cc -I. -I$SINAP_HOME/Include -I/usr/include -Wp,-H300000 -Ae
-D_HPUX_SOURCE -D_HPUX -D_LP_32_64_ +DA1.1 -L$SINAP_HOME/Library -lCASL
-o tcsend tcsend.c tcap_2.o
```

2. For HP-UX 64-bit mode:

```
cc -I. -I$SINAP_HOME/Include -I/usr/include -Wp,-H300000 -Ae
-D_HPUX_SOURCE -D_HPUX +DD64 -L$SINAP_HOME/Library/64bit -lCASL
-o tcsend tcsend.c tcap_2.o
```

3. For Solaris 32-bit mode:

```
/opt/SUNWspro/bin/cc -I. -I$SINAP_HOME/Include -I/usr/include
xtarget=ultra2 xt -DSOLARIS_SYSTEM -D_SOLARIS -D_LP_32_64_
xs xCC -L$SINAP_HOME/Library -lCASL -o tcsend tcsend.c tcap_2.o
```

4. For Solaris 64-bit mode:

```
/opt/SUNWspro/bin/cc -I. -I$SINAP_HOME/Include -I/usr/include
xtarget=ultra2 xarch=v9 xt -DSOLARIS_SYSTEM -D_SOLARIS -D__LP64__
xs xCC -L$SINAP_HOME/Library/64bit -lCASL -o tcsend tcsend.c tcap_2.o
```

5. For the Stratus ft Linux operating system:

```
cc -I. -I$SINAP_HOME/Include -I/usr/include
-DLINUX_SYSTEM -D_LINUX -DLINUX -D_LP_32_64_
-L$SINAP_HOME/Library -lCASL -lLiS -o tcsend tcsend.c tcap_2.o
```

NOTE

You must issue this command from the directory in which `tcsend.c` is located, which by default is `$SINAP_HOME/Samples/<network variant>`

Sample TCAP Application

This section describes a sample TCAP application that is made up of two test application programs (`tcsend.c` and `tcrecv.c`) that invoke another application process which you must link to the programs. Each network variant supports the sample applications described. However, the TTC variant activates these samples in a slightly different manner that is described separately.

NOTE

`tcsend` and `tcrecv` have been enhanced with a new input parameter "`y`". Currently this parameter must be used in conjunction with UDT messages and is applicable for the CCITT variant only. This parameter, when passed to the program, will verify that the data sent by `tcsend` is correctly received by `tcrecv` and vice versa. Any data mismatch will terminate the program.

`tcsend.c`

The sample program `tcsend.c` sends a TCAP message with three components to the `tcrecv.c` sample program. **Before** activating `tcsend.c`, you should activate the sample program `tcrecv.c`. Otherwise, `tcsend.c` sends messages to a nonexistent program. To activate `tcsend.c`, issue the command `tcsend` from the directory where `tcsend.c` is located (by default `$SINAP_HOME/Samples/<network variant>`).

NOTE

To avoid error conditions, do not execute multiple instances of this program at the same time. The `tcsend.c` application registers with the SINAP/SS7 system to receive control and data primitives. The SINAP/SS7 system allows only one control process per specified subsystem number (SSN); therefore, running multiple instances of this program (which would each have the same SSN) will cause problems.

`tcrecv.c`

The `tcrecv.c` sample program receives the TCAP messages sent by `tcsend.c`. To activate this sample program, issue the command `tcrecv` from the directory in which `tcrecv.c` is located (by default `$SINAP_HOME/Samples/<network variant>`). To set the debug mask for the program, enter the command `tcrecv 1`. Then run `tcsend`. If you do not execute `tcrecv` first, `tcsend.c` waits for a response from a program (`tcrecv.c`) that is not yet running.

You can run multiple instances of the `tcrecv.c` program, each of which processes an incoming MSU and sends a response back to the `tcsend.c` program. You can run up to 16 instances of `tcrecv.c`.

`tcap_2.c`

This program contains the function `print_comp`, which is called by both TCAP sample programs (`tcsend.c` and `tcrecv.c`). To run `tcsend.c` or `tcrecv.c`, you must compile the program with the `tcap_2` object file. (For instructions, see the section "Compiling the Sample Applications," earlier in this chapter.)

Sample TCAP Applications for the TTC Variant

When invoked in the TTC network variant, the `tcsend.c` and `tcrecv.c` sample programs display a series of prompts you must answer to define the program's operating characteristics (for example, local and remote SSNs, the debug mask, and the number of MSUs to send). In addition, each program displays the TTC Quality of Service (QOS) Main Menu screen, as shown in Figure 5-1.

The Quality of Service Main Menu Screen (TTC)

The TTC Quality of Service Main Menu screen is included in the series of prompts displayed by the `tcsend.c` and `tcrecv.c` programs. (The menu is included at the beginning of the `tcsend.c` prompts, and at the end of the `tcrecv.c` prompts.) The menu, shown in Figure

5-1, provides a mechanism through which you can specify values for the quality of service (QOS), priority, and sequence-control parameters.

```
TTC Quality of Service Main Menu
1. Use Default Values: Class 0 With Return, Priority 1
2. Edit Values
```

Figure 5-1. Quality of Service Main Menu Screen

To select an option from the menu, type the number corresponding to that option.

- If you type `1`, the program uses default values for the parameters.
- If you type `2`, the program prompts for a value for the QOS, priority, and sequence-control parameters. Figure 5-2 shows the series of prompts that display when you select this option. (In the figure, user input is shown in bold typeface.)

```
TTC Quality of Service Main Menu
1. Use Default Values: Class 0 With Return, Priority 1
2. Edit Values

   2

Quality of Service. Please Enter 1 - 4
1.  Connectionless Class 0 With No Return
2.  Connectionless Class 1 With No Return
3.  Connectionless Class 0 With Return
4.  Connectionless Class 1 With Return

   4

Please Enter Priority: A value of 0 - 3
   2

Please Enter Sequence Control Value: 0 - 15
   2

Quality of Service = 81
Priority = 02
Sequence Control = 02
```

Figure 5-2. The TTC Quality of Service Main Menu Screen

After you select an option from the menu and answer any prompts, the `tcsend.c` or `tcrecv.c` program displays a summary of your responses and continues processing.

The `tcrecv.c` Sample Program (TTC)

The sample program `tcrecv.c` receives MSUs from the sample program `tcsend.c` and echoes them back to the sender, `tcsend.c`. Figure 5-3 illustrates the series of prompts that `tcrecv.c` displays. Answer the prompts to define the program's operating characteristics and to select an option from the TTC Quality of Service Main Menu screen. (In the following figure, user input is shown in bold typeface.)

NOTE

You can run up to 16 instances of the `tcrecv.c` program, each of which processes an incoming MSU and sends a response back to `tcsend.c`.

```
$ tcrecv

Enter all values in decimal
local ssn(0): 3

remote ssn(0): 2

set debug mask?(0) 1

debug mask=1 LSSN=3 RSSN=2
Re-enter values? n
TC1-RECV(0010):CASL:RECV, allocating (25) in/out batch buffer
TCAP_RECV: registration is successful

TTC Quality of Service Main Menu
1. Use Default Values: Class 0 With Return, Priority 1
2. Edit Values

   1

Quality of Service = 80
Priority = 01
```

Figure 5-3. Prompts for the `tcrecv.c` Sample Program

Figure 5-4 shows the output generated by `tcrecv.c`.

NOTE

`tcrecv.c` does not generate output until it has received and processed MSUs sent by `tcsend.c`.

```
/*******************************RECV ******************************/
TC1-RECV(0010):CASL:RECV, read=1 mblocks
------->1<--------
Prim Code = TC_BEGIN Dial ID = 00 Invk id=25 prblmtype=0xfa code=0xff

------->2<--------
Prim Code = TC_INVOKE Dial ID = 00 Invk id=0 prblmtype=0x0 code=0x0

------->3<--------
Prim Code = TC_INVOKE Dial ID = 00 Invk id=1 prblmtype=0x0 code=0x0

TCAP RECV: ECHO sent(0)
TCAP RECV: ECHO sent(1)
TC1-RECV(0010):CASL:RECV, Putting msu in bb out
TC1-RECV(0010):CASL:RECV, # of outgoing msus in the batch buf=1
TCAP RECV: ECHO sent (TC_END)
TC1-RECV(0010):CASL:RECV, wrote=1 mblks to the driver
TC1-RECV(0010):CASL:RECV, read=1 mblocks
```

Figure 5-4. Output from the `tcrecv.c` Sample Program

The `tcsend.c` Sample Program (TTC)

The sample program `tcsend.c` generates and sends MSUs to the sample program `tcrecv.c`. Figure 5-5 illustrates the series of prompts that `tcsend.c` displays. Answer the prompts to select an option from the TTC Quality of Service Main Menu screen and to define the program's operating characteristics. (In the following figure, user input is shown in bold typeface.)

NOTE

You **cannot** run multiple instances of the `tcsend.c` program at the same time.

```
$ tcsend

TTC Quality of Service Main Menu
1. Use Default Values: Class 0 With Return, Priority 1
2. Edit Values

   1

Quality of Service = 80
Priority = 01

Enter all values in decimal
dpc(0): 65530
local ssn(0): 2
remote ssn(0): 3
MSUs to send before waiting for response(0): 1
Total # of MSUs to send(0)= 1
Set debug mask?(0) 1
Component timer value?(10) 5
Delay between msus?(0) 1

DPC=65530 Load gen. cnt=1 Total MSUs=1 Debug Mask=1 LSSN=2 RSSN=3
  comp timer=5 delay cnt=1
Re-enter values? n
```

Figure 5-5. Prompts for the `tcsend.c` Sample Program

Figure 5-6 shows the output generated by `tcsend.c`. After sending the specified number of MSUs, `tcsend.c` asks whether you want to send more MSUs. Specify **y** to send more MSUs. Otherwise, specify **n** to exit the program.

```
TC2-SEND(0010):CASL:SEND, allocating (25) in/out batch buffer
TCAP SEND: registration is successful
TCAP SEND: Sent comp(0)
TCAP SEND: Sent comp(1)
TC2-SEND(0010):CASL:SEND, Putting msu in bb out
TC2-SEND(0010):CASL:SEND, # of outgoing msus in the batch buf=1
TCAP SEND: Sent comp(2)
TC2-SEND(0010):CASL:SEND, wrote=1 mblks to the driver
TC2-SEND(0010):CASL:SEND, read=1 mblocks
------->1<--------
Prim Code = TC_END Dial ID = 00 Invk id=50 prblmtype=0xfa code=0xff

------->2<--------
Prim Code = TC_RESULT_L Dial ID = 00 Invk id=0 prblmtype=0x82 code=0x0

------->3<--------
Prim Code = TC_RESULT_L Dial ID = 00 Invk id=1 prblmtype=0x82 code=0x0

Send more msus?(N) n
```

Figure 5-6. Output from the `tcsend.c` Sample Program

Sample SCCP Applications

The CCITT and China network variants support sample SCCP applications. The following SCCP sample applications are located in the directory `$SINAP_HOME/Samples/<network variant>`:

- `scsend.c` creates test MSUs and sends them to the `screcv.c` program if it is running; otherwise, `scsend.c` loops the MSUs back to itself.
- `screcv.c` processes MSUs sent by `scsend.c` and prints informational messages to the terminal.

Sample MTP Applications

The CCITT and China network variants support sample MTP applications. The following MTP sample applications are located in the directory `$SINAP_HOME/Samples/<network variant>`:

- `mtpsend.c` creates test MSUs and sends them to the `mtprecv.c` program if it is running; otherwise, `mtpsend.c` loops the MSUs back to itself.
- `mtprecv.c` processes MSUs sent by `mtpsend.c` and prints informational messages to the terminal.
- `mtprx-ctl.c` is the control process for a sample application that processes test MSUs generated by an MGTS traffic generator. (Not supported by the China network variant.)
- `mtprx2.c` is a data process that `mtprx-ctl.c` initializes to process the MSUs generated by an MGTS traffic generator. (Not supported by the China network variant.)

Chapter 6
CASL Function Calls

This chapter documents the Common Application Services Layer (CASL) functions and explains how the CASL works within the SINAP/SS7 system. It contains the following sections:

- "Function Call Return Values" describes the types of errors returned by CASL functions.
- "The arch.h Include File" describes the `arch.h` (architecture) include file, which contains definitions of the structures, global variables, and enumerated data types that the SINAP/SS7 system uses for the operating system. This file is required by applications that interface with the SS7 network.
- "Common Services Functions" describes those CASL functions that are common to all types of applications (for example, the `ca_get_opc()` and `ca_register()` functions).
- "MTP and SCCP Functions" describes the CASL functions used in applications that interface with the SINAP/SS7 system at the MTP or SCCP boundary.
- "Connection-Oriented Functions" describes the CASL functions used in applications that use connection-oriented services to establish and maintain connections with other applications to exchange data.
- "TCAP Functions" describes the CASL functions used in applications that interface with the SINAP/SS7 system at the TCAP boundary.
- "IPC Functions" describes the CASL functions used by an application process to communicate with other application processes or SINAP/SS7 subsystems.
- "Load Control Functions" describes the CASL functions used in applications for implementing the load control facility.
- "BITE Functions" describes the CASL functions used in applications to implement BITE monitoring.
- "Miscellaneous Functions" describes CASL functions that can be used in any type of application.

For ISUP functions, see the *SINAP/SS7 ISDN User Part (ISUP) Guide* (R8053).

Function Call Return Values

The value returned by a CASL function call indicates whether the call was successful. To provide client application programmers with a familiar programming environment, CASL functions indicate an error condition by using the following UNIX-like methods.

- A function that normally returns a `0` or greater value will return `-1` if it is unsuccessful.
- A function that normally returns a pointer will return a `NULL` if it is unsuccessful.

In addition, the failed CASL function call sets the variable `errno` to a specific error code to indicate the reason for the failure. The include file `$SINAP_HOME/Include/ca_error.h` defines the possible error codes a CASL function can return. The UNIX file `sys/errno.h` defines UNIX error codes and their meaning.

The global memory array `CA_ERR[]` contains an ASCII string in which the first several bytes are allocated for the error number, and the remainder of the array contains a description of the error. This array, which is defined in the `$SINAP_HOME/Include/sinapintf.h` include file, is not used by all CASL functions. Note, however, that all CASL functions return `errno`.

The `arch.h` Include File

The `arch.h` include file defines the data types, enumerated types, and structures that CASL functions use, and is typically included in SINAP/SS7 application programs that interface with the SS7 network. To avoid issues associated with the length of an integer, the file defines several integer types. The data types defined in the `arch.h` include file are shown in the following example.

The include files referred to here are located in the `$SINAP_HOME/Include` directory. You might want to review these files before reading this section.

Figure 6-1. Data Types

```
#define CHAR_BITS        8       /* 8 bits in a 'char'   */

#define SHORT_BITS       16      /* 16 bits in a 'short' */

#ifdef __LP64__
#define LONG_BITS               64      /* 64 bits in a 'long'  */
#else
#define LONG_BITS               32      /* 32 bits in a 'long'  */
#endif

#define INT_BITS         32      /* 32 bits in an 'int'  */

#define ATOMIC_BITS      16      /* 16 bits in an atomic word update */

#define ARCHITECTURE     TRUE    /* Architecture defined */

#define FAR
#define NEAR

typedef char             S8;     /* 8 bit signed integer    */
typedef unsigned char    U8;     /* 8 bit unsigned integer  */

typedef short            S16;    /* 16 bit signed integer   */
typedef unsigned short   U16;    /* 16 bit unsigned integer */

typedef long             S32;    /* 32 bit signed integer   */
typedef unsigned long    U32;    /* 32 bit unsigned integer */

#ifdef __LP64__
typedef long                    S64;    /* 64 bit signed long   */
typedef unsigned long           U64;    /* 64 bit unsigned long */
#elif defined(_LP_32_64_)
typedef long long               S64;    /* 64 bit signed long   */
typedef unsigned long long      U64;    /* 64 bit unsigned long */
#endif

typedef float            F32;    /* 32 bit floating point   */

typedef double           F64;    /* 64 bit floating point   */

typedef int              BOOL;   /* Boolean value TRUE=1 or FALSE=0*/

typedef short            ATOMIC;  /* Atomic update word */
```

Common Services Functions

This section contains descriptions of the following CASL functions, which can be used by any type of application (TCAP, MTP, or SCCP).

- `ca_flush_msu()`
- `ca_get_opc()`
- `ca_register()`
- `ca_terminate()`
- `ca_withdraw()`

Descriptions of any structures and fields that these functions contain are also included.

ca_flush_msu()

SYNOPSIS

```
int     ca_flush_msu();
```

DESCRIPTION

The `ca_flush_msu()` function sends all pending outbound MSUs in the batch buffer to the SS7 driver. This function does not have any parameters.

When an application calls `ca_put_msu()` to send an MSU to the SS7 network, the MSU is placed in an output batch buffer, where it is held until the buffer becomes full. Then, the SINAP/SS7 system sends all of the MSUs in the batch buffer to the SS7 driver. You can use this function to send pending outbound MSUs to the SS7 driver without waiting for the buffer to become full.

FILES

`arch.h`, `ca_error.h`

RETURN VALUES

The `ca_flush_msu()` function returns an OPC. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful
`-1`	Unsuccessful. See `errno` for error number and description.

Possible UNIX values for `errno` are as follows.

Value	Meaning
`EBADF`	An invalid open file descriptor was specified.
`ENOTTY`	This `fides` is not associated with a device driver that accepts control functions.

Value	Meaning
EFAULT	The pointer to the specified message is outside the address space allocated to the process.
EINTR	A signal was caught during the read or system call.
EINVAL	Queue ID is not a valid message queue ID. The value of `msg_type` is less than `1`, or `msg_sz` is greater than `0` or the system-imposed limit.
EIO	An input/output (I/O) error occurred during a read or write operation.
ENXIO	The requested service cannot be performed on this particular subdevice.
ENOLINK	The link to a requested machine is no longer active.
ENOMEM	The kernel queue is full; try again.

Possible CASL values for `errno` are as follows.

Value	Meaning
CA_ERR_ACCESS	The process calling `ca_flush_msu()` is not registered. Call `ca_register()` before calling this function
CA_ERR_NO_SS7_SVC	SS7 service is not registered. Reregister the process using `ss7_primitive=SS7_CTRL_PRIMITIVE` and `fss7=1`.

This function performs a `putmsg()` and can also return the errors listed under that function.

SEE ALSO

`ca_get_msu()`, `ca_put_msu()`

ca_get_opc()

SYNOPSIS

```
int    ca_get_opc();
```

DESCRIPTION

The `ca_get_opc()` function returns a 32-bit value containing an origination point code (OPC). A client application calls this function when it needs to fill its OPC in the `T_Block` or `M_Block`. This function is applicable only to those applications registered for SS7 services.

FILES

`arch.h`, `ca_error.h`

RETURN VALUES

The `ca_get_opc()` function returns an OPC.

ca_register()

SYNOPSIS

```
int     ca_register();
```

DESCRIPTION

The `ca_register()` function registers a client application process with the SINAP/SS7 system. If the client application is composed of multiple processes, each process must register with the SINAP/SS7 system before it can use the services of the SINAP/SS7 platform.

The `ca_register()` function uses the `register_req_t` structure, which is defined in the include file `register.h`. The `register.h` include file defines the parameters that control client process registration.

NOTES

1. Before calling `ca_register()`, the client application process must initialize the global variable `CA_REG`, which contains global data used by `ca_register()`. (`CA_REG` is defined in the include file `sinapintf.h`.)

The `register_req_t` Structure

To register a client application process, you must assign values to the fields in the `register_req_t` structure, which is defined in the include file `register.h`. See the appropriate SINAP/SS7 include file(s) for definitions of any of the variables used in the `register_req_t` structure's fields (for example, `MAX_FILENAME` or `MAX_VER_SZ`).

NOTE

Differences between ANSI and CCITT field names have been eliminated in the SINAP/SS7 system. For example, the former ANSI fields have been replaced by the corresponding CCITT fields, as shown in the following chart.

Former ANSI Field	Corresponding CCITT Field
max_trans_id	max_dialogue_id
max_ism	max_invoke_id

For backward compatibility, the definitions for the ANSI field names have been maintained. However, when developing new ANSI applications, you should use the `max_dialogue_id` and `max_invoke_id` fields.

The following is an example of the `register_req_t` structure, which is defined in the include file `register.h`.

```
typedef struct register_req_s
{
      /*******************************
      *     CASL internal variables*
      *******************************/
      pid_t  pid;               /* process id - CASL puts the value*/
      pid_t  ppid;              /* parent process pid */
      int    fildes;            /* For CASL/NM use */

      /*************************************
       *      Client process identification    *
       *************************************/
      U8     node[MAX_NNAME]; /* Node name - upto 4 bytes*/
      int    filler;
      U32    lpc;
      U8     module[MAX_MNAME];  /* Module name - upto 4 bytes*/
      int    filler1;
      U8     appl[MAX_ANAME]; /* application name - upto 4 bytes*/
      int    filler2;
      U8     proc[MAX_PNAME+1]; /* Process name - upto 4 bytes */
      U8     appl_version[MAX_VER_SZ]; /* application version in ascii */
      U8     proc_version[MAX_VER_SZ]; /* process version in ascii */

      /************************
       *      IPC parameters   *
       ************************/
      BOOL    cmd_allow;             /* Flag for commands allowed from Node Mgmt */

      /************************
      *      SS#7 parameters  *
      ************************/
      BOOL    fss7;         /* Flag for communications with SS#7 io subsystem */
      U8     sio_ssn_ind;/* sio, ssn indicator */
#define REG_SIO1                /* next field is sio */
#define REG_SSN2                /* next field is ssn */
#define REG_MULT 3              /* next field is zero */
```

```
      U8      sio_ssn;                      /* contains SIO or SSN */
      U8      ss7_input_boundary;           /* type of boundary */

#define  SS7_INPUT_BOUNDARY_MTP           1
#define  SS7_INPUT_BOUNDARY_SCCP          2
#define  SS7_INPUT_BOUNDARY_TCAP          3
#define  SS7_INPUT_BOUNDARY_SCCP23        4
#define  SS7_INPUT_BOUNDARY_ISUP          5
#define  SS7_INPUT_BOUNDARY_TCAPX         6
#define  SS7_INPUT_BOUNDARY_SCCPX         7

      U8    fillerx;            /* preserve alignment */

      U32    lpc;                    /* Logical Point Code or
                                        own Point Code for process
                                        - zero defaults to own PC */

      U16     batch_count;            /* Num. of M_Blocks batched in/out */
/*ss7-1482 type of tc_count changed from S16 to S32 */
      S32     tc_count;               /* Num. of T-Blocks assigned*/
      U16   reassembly_count; /* Num of XUDT reassembly bufs */
      U8      ss7_primitive;

#define  SS7_CTRL_PRIMITIVE               1
#define  SS7_DATA_PRIMITIVE               2
#define  SS7_CTRL_DATA_PRIMITIVE          3

      U8      inbound_load_dist_type;       /* Type of load distribution*/

#define  ROUND_ROBIN                      1
#define  LEAST_UTILIZED                   2
#define  SLS_DISTRIBUTION                 3
#define  ISUP_REMOTE_SSP                  4

      S16   max_msu_input_que;        /* Maximum MSUs on input queue */
      S16   max_msu_out_que;          /* Maximum MSUs on output queue */
      S16   max_msu_holding_que; /* Maximum MSUs on holding queue */

      BOOL    fblk_que_overflow;       /* Flag - True=Block on holding queue overflow*/
      int     max_time_on_holding_que;      /* Max time in ms. for MSUs on holding queue
                                 /*before discarding*/

      /*******************************
       *  SCCP Class 2 & 3 parameters  *
       *******************************/
       U16 max_user_data_size; /* max size of user data block in bytes */
       U16 max_connections;          /* max number of connections for this   */
                                        /* process                              */
       BOOL connections_are_owned;   /* all msu's for a connection are       */
                                        /* routed to the same pid               */

      /*******************************
      *     TCAP parameters*
      *******************************/
      /* CCITT */
/*ss7-1482 type of max_dialogue_id changed from S16 to S32 */
      S32   max_dialogue_id;/* maximum number of dialogue id */
      S32   max_invoke_id;/* maximum number of invoke id */

      /* ANSI */
#define max_trans_id  max_dialogue_id   /* maximum number of transaction id */
#define max_ism max_invoke_id          /* maximum # of Invoke State Machine(ism) */

      U32   tsl_timer_value;/* value in seconds, if specified zero,
                               no transaction timer started */
      /*******************************
      *       Powerfail parameters     *
```

```
       *******************************/
       BOOL    fpf_begin;        /* Flag - Signal on pf_begin from driver*/
       BOOL    fpf_ridethru;     /* Flag - Signal on pf_ridethru from PF daemon */

       /*****************************
        *       Node Mgmt parameters           *
        *****************************/
       BOOL    fhealth_check_option;      /* Flag - To allow health check msgs */
                                         /* Next two fields define the
                                            event that will be sent on
                                            health-check timeout */
       U8     health_category;  /* Health check category */
       U8     health_subcategory;  /* Health check sub-category */
       BOOL   fparent;             /* True if the registering process
                                      is a parent process */
       BOOL   fpre_reg;            /* True - process is preregistered
                                       do not send any msgs*/

#define IPC_NOTIFY_WITHOUT_SIGNAL2/* This says send no signal
                                     but notify read() of IPC */
       /***********************
        *    BITE parameters        *
        ***********************/

       BOOL    fdebug;                    /* Flag - allow debug messages */
       BOOL    fmon_ss7;                  /* Flag - start process with ss7 monitoring on*/
       BOOL    fmon_ipc;                  /* Flag - start process with ipc monitoring on*/
       BOOL    fintercept;                /* Flag - start process with scenario running */
       U8      mon_filename[MAX_FILENAME]; /* Log file name including path
                                              if monitoring is selected*/
       U8      intc_filename[MAX_FILENAME];/* Scenario execution program name
                                              including path if intercept
                                              is selected */
       /*****************************
        *    Alarm parameters         *
        *****************************/
       U8     alarm_level;       /* Minimum Alarm Level Requested */
#define REG_NONE0
#define REG_NOTICE1
#define REG_MINOR2
#define REG_MAJOR3
#define REG_CRITICAL4
       U8     category;    /* Category of Alarms Requested */
       U8     subcategory; /* Subcategory of Alarms Requested */

       /*********************************************
        *       Application failure/recover parameters *
        *********************************************/
       U8      failure_option;          /* Action on failure */
#define  NO_ACTION                        1
#define  SEND_MESSAGE                     2
#define  SCRIPT                           3

       union
       {
             U8      script[MAX_FILENAME];   /* File name if SCRIPT */
             struct
             {                         /* Msg params if SEND_MESSAGE */
                    intmsg_type;
                   U8     node[MAX_NNAME+1];
                   U8     module[MAX_MNAME+1];
                   U8     appl[MAX_ANAME+1];
                   U8     proc[MAX_PNAME+1];
#define MAX_FDATA 24
                    U8data[MAX_FDATA];
             } msg;
       } fail;
} register_req_t;
```

* `pid` (input)
  Specifies the ID of the application process. CASL assigns this value; therefore, you should not modify this field.

* `ppid` (input)
  Specifies the ID of the parent process. CASL assigns this value; therefore, you should not modify this field.

* `fildes` (input)
  This value is for internal use (CASL and node management) and should not be modified.

* `node` (input)
  Specifies the name of the SINAP node as an ASCII string, up to four bytes long. An invalid name in this field causes the `ca_register()` function to fail. You can determine this name from the `NODE=` entry in the `/etc/sinap_master` configuration file. You must modify any script files or user-defined programs that contain an invalid, hard-coded value for the SINAP node name.

* `module` (input)
  Specifies the name of the module or system as an ASCII string, up to four bytes long. An invalid name in this field causes the `ca_register()` function to fail. You can determine this name from the `/etc/sinap_master` configuration file entry, `MODULE=`. You must modify any script files or user-defined programs that contain an invalid, hard-coded value for the SINAP module name.

* `appl` (input)
  Specifies the name of the application as an ASCII string, up to four bytes long.

  NOTE

  As part of registration, the application name becomes associated with a unique specified subsystem number (SSN). SINAP/SS7 Node Management allows only one registered application to be associated with an SSN at any time. Thus, it is important for all processes that are members of the same application to use the **same** application name at registration.

* `proc` (input)
  Specifies the name of the process as an ASCII string, up to four bytes long. Note that the names of an application's control and data processes must be different.

* `appl_version` (input)
  Specifies the version of the application as an ASCII string.

* `proc_version` (input)
  Specifies the version of the process as an ASCII string.

  NOTE

  The version information is logged and is used to check application consistency. All processes registering as members of the same application must present the same application version; however, they can present arbitrary process versions.

* `cmd_allow` (input)
  Specifies whether the client application process can receive MML commands through the IPC queue. Use `1` to allow the process to receive MML commands in this manner; otherwise, use `0`.

* `fss7` (input)
  Specifies whether the client application process requires SS7 services. Use `1` to indicate that the client application requires SS7 services. In this case, the remaining SS7 parameters are evaluated to determine the type of SS7 services that will be supplied. Use `0` to indicate that the client application does not require SS7 services. In this case, the fields up to `max_time_on_holding_que` are ignored.

* `sio_ssn_ind` (input)
  Specifies whether a service information octet (SIO) or SSN is supplied in the `sio_ssn` field. Use `1` to supply an SIO; use `2` to supply an SSN; use `3` to implement enhanced message distribution (see the section "Enhanced Message Distribution" in Chapter 3). The value you specify for the `ss7_input_boundary` parameter determines whether an SIO or SSN is required.

* `sio_ssn` (input)
  Specifies the SIO or SSN that the SINAP/SS7 system is to associate with the process. The SINAP/SS7 system uses this SIO or SSN to identify the process; therefore, you must specify a unique value for each process. If you are using an SIO, specify a value in the range `1` to `15`; if you are using an SSN, specify a value in the range `2` to `255`. If an application consists of multiple processes, each process must specify the same SIO or SSN.

  If you specified `3` for `sio_ssn_ind`, specify zero for this field and use the `dist_cmd_t` structure to define the SSN(s) to be associated with the application process.

* `ss7_input_boundary` (input)
  Specifies the boundary at which the application receives communication from the SS7 network. To receive input at one of the following boundaries, specify the code associated with the boundary.

To Define SS7 Input Boundary Type	Enter
MTP	1
SCCP	2
TCAP	3
SCCP23 (Connection Oriented Services Classes 2 & 3)	4
ISUP	5
TCAPX (supports XUDT)	6
SCCPX (supports XUDT)	7

  If you specify MTP or ISUP as the input boundary, you must specify an SIO code in the `sio_ssn` field. If you specify SCCP or TCAP, you must specify an SSN code in the `sio_ssn` field unless you are implementing enhanced message distribution, in which case the `sio_ssn` field is zero.

* `fillerx` (input)
  Functions as a filler to preserve the alignment.

* `lpc` (input)
  Specifies the SINAP node's own signaling point code (OSP) if the field contains a value of 0 (the default value). If the Distributed Logical Point Code (DLPC) feature is configured on the SINAP node, this field can also specify the logical point code (LPC) of an ISUP application process. For detailed information on the DLPC feature, see the *SINAP/SS7 ISDN User Part (ISUP) Guide* (R8053).

* `batch_count` (input)
  Specifies the number of `M_Blocks` (MSUs) the process will transfer to or from the SS7 driver with a single call. If you specify a large value for this field, you increase the SINAP/SS7 efficiency; however, if you find that the average processing delay per `M_Block` is large, you should specify a smaller value for this field.

  All outbound MSUs are held in a batch buffer until the buffer becomes full. Then, the SINAP/SS7 system flushes all of the MSUs to the SS7 SVR4 driver. For receiving MSUs, if there are no buffers pending in the batch buffer, the SINAP/SS7 system returns the

`M_Block` address. Additionally, if there are no inbound MSUs pending on a read, the SINAP/SS7 system flushes the output batch buffer.

* `tc_count` (input)
  Specifies the number of `T_Blocks` the CASL will allocate in the `T_Block` array. This parameter applies only to client application processes registered to receive input at the TCAP boundary.

* `reassembly_count` (input)
  Specifies the number of XUDT reassembly buffers allocated.

* `ss7_primitive` (input)
  Specifies whether the client application process receives control or data primitives, or both.

Specify	To Indicate
1	the client application process will receive control primitives
2	the client application process will receive data primitives
3	the client application process will receive both control and data primitives

If more than one process is registered for data primitives, the rules for load distribution apply (see the description of `inbound_load_dist_type`). For each SSN, only one process can be registered to receive control primitives and that process must set the `fss7` field to `FALSE`. Each SSN can have up to 16 processes registered to receive data primitives.

* `inbound_load_dist_type` (input)
  Specifies the type of load distribution policy to use.

Specify	To Indicate
1	`ROUND_ROBIN`, which distributes MSUs evenly across each of the application's input queues.
2	`LEAST_UTILIZED`, which places each arriving MSU in the queue with the least backlog.

Specify	To Indicate
3	`SLS_DISTRIBUTION`, which places each MSU in the application queue associated with the link over which the MSU was routed.

* `max_msu_input_que` (input)
  Specifies the maximum number of `M_Blocks` that the SINAP/SS7 system can store pending a client application read. If this number is exceeded, the driver discards any incoming `M_Blocks`. Select a number that is a small multiple of the blocking factor (which is the `batch_count` variable described previously in this section) and that can account for the way the client application receives several messages and processes them in one "burst." The MSU input queue size allows a minimum value of 7000 through a maximum value of 32000. This value is defined in the `sinap.h` include file.

* `max_msu_out_que` (input)
  Specifies the maximum number of `M_Blocks` that the SINAP/SS7 system can store pending a client application process write. The output limit should complement the input limit.

* `max_msu_holding_que` (input)
  Specifies the maximum number of `M_Blocks` that can be stored on a holding queue before an overflow condition occurs. A *holding queue* is temporary queue that the SINAP/SS7 system creates when routing to a particular destination (or set of destinations) is temporarily blocked (for example, during a link changeover). When routing is resolved, the SINAP/SS7 system takes the `M_Blocks` from the holding queue and routes them to their destinations.

* `fblk_queue_overflow` (input)
  Specifies how the SINAP/SS7 system treats a holding queue overflow condition. If set to `1`, an `M_Block` send request exceeding the maximum specified in `max_msu_holding_que` causes that call to block pending route resolution. If set to `0`, the sender process is notified of the condition through an error return.

* `max_time_on_holding_que` (input)
  Specifies the length of time (in milliseconds) that `M_Blocks` remain on the holding queue before being discarded.

* `max_user_data_size` (input)
  Specifies the maximum size, in bytes, of the user data block.

* `max_connections` (input)
  Specifies the maximum number of connections allowed for this process.

* `connections_are_owned` (input)
  Specifies that all MSUs for a connection are routed to the same PID.

* `max_dialogue_id` (input)
  (CCITT) Specifies the maximum number of dialogue IDs.

* `max_invoke_id` (input)
  (CCITT) Specifies the maximum number of invoke IDs.

* `max_trans_id` (input)
  (ANSI) Specifies the maximum number of transaction IDs. This field description is maintained for backward compatibility only. For developing new ANSI applications, use the `max_dialogue_id` field instead of this field.

* `max_ism` (input)
  (ANSI) Specifies the maximum number of invoke state machines.This field description is maintained for backward compatibility only. For developing new ANSI applications, use the `max_invoke_id` field instead of this field.

* `tsl_timer_value` (input)
  Specifies the value, in seconds, of the transaction timer. This timer times receipt of a message. If the transaction timer expires, an alarm is sent to Node Management and the pending transaction is aborted. The value of this timer should be less than the setting of the environment variable `TCAP_TQ_BINS` or the default value of half the sum of `MIN_TQ_BINS` and `MAX_TQ_BINS` which are defined in `tcglob.h`. For example: (4 + 3601)/2 = 1802. Use `0` to indicate that the transaction timer should not be used.

* `fpf_begin` (input)
  Specifies whether the client application process is signaled when the SINAP/SS7 system detects a `pf_begin` (power-fail) event. When this signal occurs, the client application has five seconds of guaranteed operation before failure. In this interval, the application can coordinate with its mate or withdraw from the SS7 network. Use `1` to specify that the process should be signaled; otherwise, use `0`.

* `fpf_ridethru` (input)
  Specifies whether the client application process is signaled when the SINAP/SS7 system detects a `pf_ridethru` (power-fail ride-through) event.

* `fhealth_check_option` (input)
  Specifies whether Node Management sends periodic health-check messages to the client application process by means of IPC. Use `1` to specify that node management should send health-check messages to the application; otherwise, use `0`. (The SINAP/SS7 environment variable `SINAP_HEALTH_INTERVAL` specifies the frequency with which the SINAP/SS7 system sends health-check messages; see the *SINAP/SS7 User's Guide* (R8051) for more information.)

The destination process must respond within the amount of time specified by the SINAP/SS7 environment variable `SINAP_HEALTH_TIMEOUT` (see the *SINAP/SS7 User's Guide* (R8051) for more information about SINAP/SS7 environment variables). If two successive health-check requests fail, Node Management declares the process failed and invokes the actions specified in the `health_category` and `health_subcategory` fields.

* `health_category` (input)
  Specifies the category of actions that Node Management takes when two consecutive health-check requests fail.

* `health_subcategory` (input)
  Specifies the subcategory of actions that Node Management takes when two consecutive health-check requests fail.

* `fparent` (input)
  Specifies whether the client application process is a parent process. Use `1` to indicate that the application process is a parent; otherwise use `0`.

* `fpre_reg` (input)
  This field is internal to the SINAP/SS7 system; you should not modify it.

* `fsignal` (input)
  Specifies whether the client application process is signaled for incoming IPC messages. Use `1` if you want the SINAP/SS7 system to generate a signal when the process receives an IPC message; otherwise, use `0`. If you specify `1`, you must also initialize the signal `SIG_S7_IPC` with your application's signal-handler process.

  Use `2` (`IPC_NOTIFY_WITHOUT_SIGNAL`) if you want the SINAP/SS7 system to cause a *blocking-mode* call to `ca_get_msu()` or `ca_get_tc()` to return with `errno=EINTR` when an IPC message arrives for the application process. (For more information about handling such a return, see the section "Error Handling" in Chapter 3.)

* `fdebug` (input)
  Specifies whether the client application process receives BITE debug messages. Use `1` to indicate that the application permits debug messages. You may find it useful to set this field according to a command-line parameter.

* `fmon_ss7` (input)
  Specifies that when the application process is started, the SINAP/SS7 system is to automatically initiate a BITE monitor process to monitor and log all of the SS7 activities for the application process. Use `1` to initiate the BITE monitor process; otherwise, use `0`. If you use `1`, be sure to provide a value for the `mon_filename` field.

* `fmon_ipc` (input)
  Specifies that when the application process is started, the SINAP/SS7 system is to automatically initiate a BITE monitor process to monitor and log all of the IPC activities

for the application process. Use `1` to initiate the BITE monitor process; otherwise, use `0`. If you use `1`, be sure to provide a value for the `mon_filename` field.

* `fintercept` (input)
  Specifies whether the SINAP/SS7 system intercepts all SS7 traffic and passes it to the BITE for scenario execution. Use `1` to enable intercept mode; otherwise, use `0`. This field allows an application process to start in intercept mode.

  NOTE

  You may find it useful to set this field according to a command-line parameter. You can also enable scenario execution by using the MML commands `START_SCEN` and `STOP_SCEN` (see Appendix A).

* `mon_filename` (input)
  Specifies the path name of the log file to which BITE monitoring messages are to be written. This field applies only if you have enabled BITE monitoring by specifying the value `1` for the `fmon_ss7` or `fmon_ipc` field.

  NOTE

  If you enable BITE monitoring but do not specify a value for this field, `ca_register()` returns an error.

* `intc_filename` (input)
  Specifies the name of the scenario execution program that is to be executed whenever the SINAP/SS7 system intercepts SS7 traffic.

  NOTE

  This field applies only if you have specified intercept mode (see the description of `fintercept` earlier in this section).

* `alarm_level` (input)
  Specifies the minimum alarm level that will be sent to the client application process. Use `4` to indicate critical alarms, `3` to indicate major alarms, `2` to indicate minor alarms, `1` to indicate a notice, and `0` to indicate no alarms.

* `category` (input)
  Specifies the classification of a set of alarms. A category can be established for any set of alarms that share the same subcategories. Categories are defined for the system; each value for the `category` parameter has a unique meaning to the system. Use any number from `15` through `30`.

* `subcategory` (input)
  Specifies the classification of an alarm within a category. The client processes sharing the alarm `category` must make the subcategory assignments and reservations. You can specify up to 30 subcategories.

* `failure_option` (input)
  Specifies what action, if any, the SINAP/SS7 system takes if the registering process fails. Specify `1` if the SINAP/SS7 system should not take action, specify `2` if the SINAP/SS7 system should send a message by means of the IPC, or specify `3` if the SINAP/SS7 system should execute a script.

  If you specify `3` for `failure_option`, specify the script name in the `script` field.

  - `script` (input)

    Specifies the name of the script file the SINAP/SS7 system will execute if the client process fails.

  If you use `2` as the value for `failure_option`, specify the message and destination in the following fields.

  - `msg_type` (input)

    Specifies the type of message to send if the client application process fails.

  - `node` (input)

    Specifies the name of the node to which the message is being sent. The value of `node` can be up to four bytes long.

  - `module` (input)

    Specifies the name of the module to which the message is being sent. The value of `module` can be up to four bytes long.

  - `appl` (input)

    Specifies the name of the application to which the message is being sent. The value of `appl` can be up to four bytes long.

  - `proc` (input)

    Specifies the name of the process to which the message is being sent. The value of `proc` can be up to four bytes long.

  - `data` (input)

    Specifies the message to send to the client application process if it fails. The value of `data` can be up to 24 bytes long.

FILES

`arch.h`, `ca_error.h`, `iblock.h`, `register.h`, `sinap.h`, `sinapintf.h`, `sysdefs.h`

RETURN VALUES

The `ca_register()` function can return the following values. If `ca_register()` returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

NOTE

The `ca_register()` function does not close the SINAP file descriptor in all cases of an error return. Attempts to retry `ca_register()` in such cases, will always fail with `errno 16=EBUSY`. Before attempting a retry, `(void) close(CA_FD)` ignoring any error return.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

Possible UNIX values for `errno` are as follows.

Value	Meaning
`EBADF`	An invalid open file descriptor was specified.
`EBUSY`	Indicates that the mount device is busy and the system cannot start the operation.
`EEXIST`	`O_CREAT` and `OEXCL` are set and the named device exists.
`EFAULT`	The specified path name points outside this process's allocated address space.
`EINTR`	The signal was caught during the `open()` system call.
`EIO`	An I/O error occurred during a read or write operation.
`EISDIR`	The named file is a directory and the `oflag` is `write` or `read/write`.
`EMFILE`	`NOFILES` file descriptors are currently open.
`ENFILE`	The system table is full.
`ENOLINK`	The link to a requested machine is no longer active.

Value	Meaning
ENOSPC	O_CREAT and OEXCL are set and the file system is out of I-nodes. The device does not exist; O_CREAT is specified.
ENOTDIR	A component of the path prefix is not a directory.
ENOTTY	This `fides` is not associated with a device driver that accepts control functions.
EROFS	The named file resides on a read-only file system and `oflag` is `write` or `read`/`write`.
ENXIO	The requested service cannot be performed on this particular subdevice.

Possible CASL errors are as follows.

Value	Meaning
CA_ERR_ACCESS	The process is not registered. Call `ca_register()` before calling this function.
CA_ERR_ALREADY_REG	The process is registered more than once.
CA_ERR_INT_MML	There is an error in a command to the BITE.
CA_ERR_INVALID_PUT_REQ	The `ss7_input_boundry` is invalid for XUDT message processing.
CA_ERR_REG_BCNT_HIGH	The specified value for the batch buffer count is invalid.
CA_ERR_REG_FAILURE_OPT	The specified value for failure options is invalid.
CA_ERR_REG_INTC_FILE	The scenario execution file is missing.
CA_ERR_REG_LOAD_DIST	The specified value for the load distribution type is invalid.
CA_ERR_REG_LPC_INVALID	The process is not registered at the ISUP boundary. This error can only occur if the Distributed Logical Point Code (DLPC) feature is configured.
CA_ERR_REG_MAX_HOLD	The specified value for the maximum hold buffer count is invalid.
CA_ERR_REG_MAX_INMSU	The specified value for the maximum input MSU count is invalid.

Value	Meaning
CA_ERR_REG_MAX_OUTMSU	The specified value for the maximum output MSU count is invalid.
CA_ERR_REG_MAX_TIME	The maximum time value is missing.
CA_ERR_REG_MON_FILE	The monitor log file name is missing.
CA_ERR_REG_NORESP	There is no response from client management.
CA_ERR_REG_SCR_FILE	The script file name is missing.
CA_ERR_REG_SIO	The specified value for the SIO is invalid.
CA_ERR_REG_SIO_SSN_IND	The specified value for the SSN/SIO indicator is invalid.
CA_ERR_REG_SS7_BOUND	The specified value for the SS7 input boundary is invalid.
CA_ERR_REG_SS7_PRIMITIVE	The specified value for the SS7 primitive is invalid.
CA_ERR_REG_SSN	The specified value for the SSN is invalid.
CA_ERR_REG_TCCOUNT	The specified value for the TC count is invalid.

If the TCAP returns the error TC_ERR_INV_TCAP_REG_PARAMETERS, correct the TCAP registration parameters: tc_count, max_trans_id, and max_ism. Then reregister the application process.

This function performs a ca_put_msg(), a ca_get_msg(), and a ca_get_key(), and can also return the errors listed under those functions.

SEE ALSO

ca_terminate()

ca_terminate()

SYNOPSIS

```
int     ca_terminate(
          terminate_t   *pterm);
```

DESCRIPTION

The `ca_terminate()` function terminates an application process and deallocates its IPC and SS7 resources (such as table entries, structure entries, and shared memory resources). Any process registered with the SINAP/SS7 system can call `ca_terminate()` to terminate itself. A parent process can also issue this function on behalf of its child processes.

After an application process calls the `ca_terminate()` function, all actions previously performed by `ca_register()` are undone. The application process is no longer registered with the SINAP/SS7 system and can no longer receive MSUs or IPCs. The application process is then free to reregister with another call to `ca_register()` using different parameters, if required, or to call the `exit()` function.

PARAMETERS

* `pterm` (input)
  Specifies a pointer to the terminate structure, `terminate_t`, which is defined in the include file `terminate.h`. For more information, see the following section, "The terminate_t Structure."

The `terminate_t` Structure

Before calling the `ca_terminate()` function, you must assign values to the following fields in the `terminate_t` structure.

```
typedef struct  terminate_s
{
        ipc_key_t       ipc_key;
        U8      msg_type;
        BOOL    fss;
        U8      reason[80];
        int     exit_code;
} terminate_t;
```

* `ipc_key` (input)
  Specifies the IPC key of the process to be terminated. The IPC key is defined by the `ipc_key_t` structure, which is described in the following section, "The IPC Key Structure (ipc_key_t)."

* `msg_type` (input)
  Specifies how the process is to be terminated.

  Use the value `TERM_COMMANDED` (or `1`) to specify self-commanded. With this value, the parent process kills the child process specified by `ipc_key`.

  Use the value `TERM_SELF_INITIATED` (or `2`) to specify self-initiated. This value disconnects a process from the SINAP/SS7 system, but not exit.

* `fss` (input)
  Specifies whether the SINAP/SS7 system is to terminate the parent process and its child processes, or just this process. Use the value `TRUE` (or `1`) to terminate the parent process and all of its child processes; use `FALSE` (or `0`) to terminate this process only.

* `reason` (input)
  Specifies the reason for terminating the process. This field can be up to 80 bytes long.

* `exit_code` (input)
  This field is unused.

The IPC Key Structure (`ipc_key_t`)

The `ipc_key_t` structure is defined in the include file `sinap.h` and has the following format.

The `ca_ascii_u32()` function assigns values to the `node`, `module`, `appl`, and `proc` fields of the `ipc_key_t` structure, which is shown below.

```
typedef struct ipc_key_s
{
        U32      node;
        U32      module;
        U32      appl;
        U32      proc;
        U8       instance;
        U8       node_index;
        U16      ipc_index;
} ipc_key_t;
```

* `node` (input)
  Specifies the ID of the SINAP node on which the application is running. You can determine this value from the `NODE=` entry in the `/etc/sinap_master` file. You should modify any script files or user-defined programs that contain an invalid, hard-coded node name.

* `module` (input)
  Specifies the name or ID of the module. You can determine this value from the `MODULE=` entry in the `/etc/sinap_master` file. You should modify any script files or user-defined programs that contain an invalid, hard-coded module name.

* `appl` (input)
  Specifies the compressed application ID.

* `proc` (input)
  Specifies the compressed process ID.

* `instance` (input)
  Specifies the instance ID (a value in the range `1` through `16`). A value of `0` indicates that the field is not used.

* `node_index` (input)
  Specifies the index value (in the range 0 through 3) of the node. This value is equal to the values of the node ID and the SINAP_INDEX environment variable. This value is internal and should not be changed.

* `ipc_index` (input)
  Specifies the index ID of the IPC process table.

FILES

`arch.h`, `ca_error.h`, `sinap.h`, `terminate.h`

RETURN VALUES

The `ca_terminate()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

Possible CASL errors are as follows.

Value	Meaning
CA_ERR_ACCESS	The process calling `ca_terminate()` is not registered. Call `ca_register()` before calling this function.
CA_ERR_DESTN_KEY	Destination process key not found.
CA_ERR_IBLK_DATA	The `I_Block` data exceeds the maximum limit.
CA_ERR_IBLK_MSGTYPE	Invalid message type.

This function performs a `ca_get_msg()` and a `ca_put_msg()` and can also return the errors listed under those functions.

SEE ALSO

`ca_register()`

ca_withdraw()

SYNOPSIS

```
int     ca_withdraw();
```

DESCRIPTION

The `ca_withdraw()` function allows an orderly withdrawal from SS7 service by removing a client application instance from the SINAP/SS7 input load distribution. This prevents new messages from being routed to that client application process, while allowing the application process to continue processing existing messages.

Though transactions that are currently being routed to that application process continue to be processed, load distribution does not route any new transactions. A withdrawn client application can initiate transactions. Continuation of these transactions is routed to the withdrawn application instance. When all transactions are completed, the application instance can perform a `ca_terminate()`.

FILES

`arch.h`, `ca_error.h`

RETURN VALUES

The `ca_withdraw()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning. See `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

Possible UNIX values for `errno` are as follows.

Value	Meaning
`EBADF`	An invalid open file descriptor was specified.
`ENOTTY`	This `fides` is not associated with a device driver that accepts control functions.
`EFAULT`	The specified path name points outside this process's allocated address space.
`EINVAL`	Queue ID is not a valid message queue ID. The value of `msg_type` is less than `1`, or `msg_sz` is less than `0` or greater than the system-imposed limit.
`EINTR`	The signal was caught during the `open()` system call.
`ENXIO`	The requested service cannot be performed on this particular subdevice.
`EIO`	An I/O error occurred during a read or write operation.
`ENOLINK`	The link to a requested machine is no longer active.

A possible CASL error follows.

Value	Meaning
`CA_ERR_ACCESS`	The process calling `ca_withdraw()` is not registered. Call `ca_register()` before calling this function.

SEE ALSO

`ca_terminate()`

MTP and SCCP Functions

This section contains an alphabetic reference of the following CASL functions, which are used in applications that interface to the SS7 network at the MTP or SCCP boundary.

- `ca_get_msu()`
- `ca_get_msu_noxudt()`
- `ca_lookup_gt()`
- `ca_put_msu()`

The structures and fields for each function are also described in this section. The function descriptions apply to all network variants of the SINAP/SS7 system; any differences are noted.

ca_get_msu()

SYNOPSIS

```
m_block_t    *ca_get_msu(
                BOOL    fwait);
```

DESCRIPTION

The `ca_get_msu()` function returns a pointer to an inbound MSU. A client application registered to receive input at the MTP or SCCP boundary can call this function.

The input batch buffer is the first source for inbound `M_Block` requests. The SINAP/SS7 system queries the SS7 driver for up to the maximum number of `M_Blocks` the batch buffer can hold. When the SINAP/SS7 system completes the query, `ca_get_msu()` returns a pointer to the first `M_Block` in the batch buffer. If the batch buffer is not empty, `ca_get_msu()` returns a pointer to the next `M_Block`.

CAUTION

The client application must allocate enough inbound and outbound batch buffer space (by means of the `max_msu_input_que` and `max_msu_output_que` parameters of the `ca_register()` function.) If there is not enough inbound batch buffer space, consecutive calls to `ca_get_msu()` can destroy the previous contents of the `M_Block` (MSU). The client application should either copy the MSU or allocate enough buffers during registration.

If the results of the query to the SS7 driver indicate that there are no `M_Blocks` currently pending in the SS7 driver for the client process, `ca_get_msu()` ensures that any partial outbound batches are cleared.

NOTE

Because of the UNIX signal, it is possible that when this function is called with the `fwait` parameter set to `1`, the function might return an error, indicating that the signal was

invoked. If this happens, the calling process must issue the call again, or dequeue any IPC messages.

PARAMETERS

* `fwait` (input)
  Specifies whether the function is to wait for an MSU. Use `1` to indicate that the function is to wait for an MSU; otherwise, use `0`. If you use `0`, the function returns the error `ENODATA` when there are no MSUs.

The Main `M_Block` Structure (`m_block_t`)

The following fields make up the `m_block_t` structure, which is defined in the include file `mblock.h`.

```
 typedef  struct  m_block_s
{
        ca_ctrl_t         ca_ctrl;
        timestamp_t       ts;
        bi_ctrl_t         bi_ctrl;
        tcap_ctrl_t       tc_ctrl;
        sccp_ctrl_t       sc_ctrl;
        sccp_prim_t       sc_prim;
        tcap_alt_t        tc_alt;
        mtp_ctrl_t        mtp_ctrl;
        union m_block_ud_tag
        {
          msu_t           msu;
          ccitt_msu_t     ccitt_msu;    /* CCITT MSU Data */
          ttc_msu_t       ttc_msu;     /* TTC and NTT MSU Data */
          ansi_msu_t      ansi_msu;  /* ANSI and China MSU Data */
          13_event_t      dr_event;
          iblk_t          ib;
        } ud;
} m_block_t;
```

* `ca_ctrl` (output)
  Specifies CASL control information. For information about this structure's fields, see "The CASL Control Structure (ca_ctrl_t)" later in this section.

* `ts` (output)
  Specifies a collection of timestamps that the SINAP/SS7 system automatically inserts. The timestamps aid monitoring and logging and are visible when you run the BITE log-analysis program. For information about this structure's fields, see "The Timestamp Structure (timestamp_t)" later in this section.

* `bi_ctrl` (output)
  Specifies BITE control information. For information about this structure's fields, see "The BITE Control Structure (bi_ctrl_t)" later in this section.

* `tc_ctrl` (output)
  Specifies TCAP control information. For information about this structure's fields, see "The TCAP Control Structure (tcap_ctrl_t)" later in this section.

* `sc_ctrl` (output)
  Specifies SCCP control information. For information about this structure's fields, see "The SCCP Control Structure (sccp_ctrl_t)" later in this section.

* `sc_prim` (output)
  Specifies the `sccp_prim_t` structure, which conveys information about large messages. This structure is defined in the `Mblock.h` include file. See "The sccp_prim_t Structure" later in this chapter for more information.

* `tc_alt` (output)
  The SINAP/SS7 system uses this field for specifying the alternative DPC (refer to chapter 3) for the outbound MSU, i.e. refer to ca_put_msu() later in this chapter. You should not modify it at ca_get_msu(), which is for the inbound MSU.

* `mtp_ctrl` (output)
  Specifies MTP control information. For information about this structure's fields, see "The MTP Control Structure (mtp_ctrl_t)" later in this section.

* `ud` (output)
  Specifies the union of `M_Block`.

  - `msu` (output)
    Specifies user data for the MSU or `I_Block` information. See "The MSU Data Structure (msu_t)" later in this section for information about this structure's fields.
  - `dr_event` (output)
    This field is internal to the SINAP/SS7 system; you should not modify it. For information about this structure's fields, see "The l3_event_t Structure" later in this section.
  - `ib` (output)
    See "The iblk_t Structure" later in this section for information about this structure's fields.
  - `ccitt_msu` (output)
    Specifies MSU data for the CCITT network variant.
  - `ttc_msu` (output)
    Specifies MSU data for the TTC and NTT network variants.
  - `ansi_msu` (output)
    Specifies MSU data for the ANSI and China network variants.

The CASL Control Structure (`ca_ctrl_t`)

The following fields make up the `ca_ctrl_t` structure, which is defined in the include file `blkhdr.h`.

```
typedef struct   ca_ctrl_s
{
       int     msg_type;       /* For compatibility with the existing UNIX IPC
                                      mechanism. */
       int     msu_cnt;        /* # of MSUs pending */
       int     free_cnt;       /* # of free MSUs in read queue */
       int     wfree_cnt;      /* # of free MSUs in write queue */
       S16     lost_cnt;       /* # of MSUs lost due to insuff. resources */
       S16     data_size;      /* Total size of structure excluding this
                                  structure. */
       U8      node_index;     /* index (0 - 3) of current node */
       U8      sinap_variant;  /* V_CCITT, V_ANSI, V_HYBRID, V_TTC */
       U16     link;           /* index of the origination link */
       pid_t   pid;            /* Process ID of a specific process or 0
                                  for load distribution */
       int     msg_sender;     /* Set to 0 if from link otherwise contains
                                  the process ID */
       U8      iblk;           /* TRUE if data contains I_Block */
                               /* Not used anywhere!!!!! */
       U8      rw;             /* Flag for monitor message only */
       U8      monitor_id;     /* monitor ID for monitored MSU  */
       U8      ssn_sio;        /* SSN or SIO ID for load distribution. */
       struct {
               U32     timer_id;       /* For CASL internal use only */
               U32     timer_val;      /* For CASL internal use only */
               int     omsg_type;      /* For CASL internal use only */
       } timr;
       struct {                        /* For internal use only */
               int     source;         /* For distribution managment. */
               int     destination;    /* For protocol processing. */
#ifdef _KERNEL
               mblk_t  *mptr;          /* Back pointer to mblk_t.  L3 only.*/
#else
               void    *mptr;          /* ss7-1102 - dummy back pointer. */
#ifdef _LP_32_64_
               U32  filler;            /* For User32/Driver64 compatibility*/
#endif  /* _LP_32_64_ */
#endif  /* _KERNEL */
       } internal;
} ca_ctrl_t;
```

* `msg_type` (output)
  Specifies the type of MSU being sent. This field is compatible with the existing UNIX IPC mechanism.

* `data_size` (output)
  Specifies the total size of the structure, excluding this field.

* `node_index` (output)
  This is an internal field that is automatically initialized to the appropriate value for the SINAP node.

* `sinap_variant` (output)
  This is an internal field that is automatically initialized to the appropriate value for the network variant being used on the SINAP node.

* `lost_cnt` (output)
  Specifies the number of `M_Blocks` lost due to insufficient resources within the SS7 driver.

* `msu_cnt` (output)
  Specifies the number of MSUs pending.

* `free_cnt` (output)
  Specifies the number of free MSUs pending in the read queue.

* `wfree_cnt` (output)
  Specifies the number of free MSUs pending in the write queue.

The following fields are internal to the SINAP/SS7 system and you should not modify them:

* `pid` (output)
* `link` (output)
* `msg_sender` (output)
* `iblk` (output)
* `rw` (output)
* `monitor_id` (output)
* `ssn_sio` (output)
* `source` (output)
* `destination` (output)
* `mptr` (input) Specifies a pointer to `m_block_t`, Level 3.
* `timer_id` (output)
* `timer_val` (output)
* `omsg_type` (output)

The Timestamp Structure (`timestamp_t`)

The following fields make up the `timestamp_t` structure, which is defined in the include file `timestamp.h`.

```
typedef struct timestamp_s
{
         U16     index;
         stamp_t stamp[MAX_TIME_STAMPS];
}timestamp_t;
```

* `index` (output)
  Specifies the next slot to stamp the time.

* `stamp[MAX_TIME_STAMPS]` (output)
  Specifies the timestamp slots. For an explanation, see "The stamp_t Structure." (`MAX_TIME_SLOTS` is defined in the SINAP/SS7 `timestamp.h` include file.)

The `stamp_t` Structure

The following fields make up the `stamp_t` structure, which is defined in the include file `timestamp.h`.

```
typedef struct stamp_s
{
       U32     secs;                   /* time in seconds since 1/1/70 */
       U8      tsid;                   /* timestamp id                 */
       U8      ipcx;                   /* ipc index if applicable      */
       U16     msec;                   /* time in milliseconds         */
} stamp_t;
```

* `secs` (output)
  Specifies the time (in seconds) since 1/1/70.

* `tsid` (output)
  Specifies the timestamp ID. Definitions for these IDs can be found in the include file `timestamp.h`.

* `ipcx` (output)
  Specifies the IPC index, if applicable.

* `msec` (output)
  Specifies the time, in milliseconds.

The BITE Control Structure (`bi_ctrl_t`)

The following fields make up the `bi_ctrl_t` structure, which is defined in the include file `mblock.h`.

```
typedef struct  bi_ctrl_s
{
       U8      command;                /* Command type */
       U8      qualifier;              /* command qualifier*/
       U8      rw;                     /* monitor read/write/both */
       U8      monitor_id;
       U8      filler[2];              /* For User32/Driver64 compatibility*/
       U16     link;                   /* link index */
       pid_t   pid[2];                 /* application and SE process IDs*/
} bi_ctrl_t;
```

These fields are internal to the SINAP/SS7 system and you should not modify them.

* `command` (output)
* `qualifier` (output)
* `link` (output) - The UNIX include file `sys/types.h` defines the `dev_t` structure.
* `pid[2]` (output)
* `rw` (output)
* `monitor_id` (output)

The TCAP Control Structure (`tcap_ctrl_t`)

The following fields make up the `tcap_ctrl_t` structure, which is defined in the include file `mblock.h`.

```
typedef struct    tcap_ctrl_s
{
        S16     ipc_index;
        S32     trans_id;
        U8      tcap_msg_type;
        U8      abort_type;
        U8      abort_cause;
        U8      call_disposition;      /* fictitious OPC */
} tcap_ctrl_t;
```

These fields are internal to the SINAP/SS7 system and you should not modify them.

* `ipc_index` (output)
* `trans_id` (output)
* `tcap_msg_type` (output)
* `abort_type` (output)

* `abort_cause` (output)

The following field must be specified for the ANSI network variant only:

* `call_disposition` (output)
  (ANSI) This field indicates the FOPC feature is to be used in the MTP header. You should not modify this field.

The SCCP Control Structure (`sccp_ctrl_t`)

The `sccp_ctrl_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct    sccp_ctrl_s
{
       U8     sccp_ctrl;
       U8     sccp_source;
       U8     sccp_dest;
       U8     ccitt_sls5;
       U16    sccp_err_no;
       U8     sccp_msg_priority;
       U8     sccp_seq_control;
       U8     sccp_3rd_party_addr[MAX_ADDR_LEN];
       U8     importance_parm;
       U8     seg_included;
       U8     filler1;
       U8     filler2;
} sccp_ctrl_t;
```

* `sccp_ctrl` (output)
  This field is meaningful to the SCCP user. SCCP Management sets this field to `N-UNITDATA` or `N-NOTICE`.

* `sccp_source` (output)
  This field specifies which SCCP function originated the message. You should not modify it.

* `sccp_dest` (output)
  This field specifies the SCCP function for which the message is destined. You should not modify it.

* `ccitt_sls5` (output)
  (CCITT) This field specifies the five-bit SLS field, used only in the CCITT network variant. Bit 4 of the 5-bit SLS is used to randomly choose the route/link set (0 or 1) *if* the route set (RSET) is configured with two route/link sets for load sharing *and* the SLS value of the MSU label is mapped to a corresponding physical link in that route/link set over which to send the outbound message. This field is used only to send outbound MSUs for MTP, SCCP, or TCAP applications. The field is not used to determine the route/link set that

is used for inbound SS7 messages or the outbound SLT or SNM messages. You should not modify this field.

* `sccp_err_no` (output)
  This field indicates the SCCP error. You should not modify it.

* `sccp_msg_priority` (output)
  This field specifies the message priority for the MSU. This priority parameter is valid only for SCCP Class 0 and Class 1 messages. Valid values are `0` through `3` (lowest to highest priority, respectively).

* `sccp_seq_control` (output)
  This field specifies the value to use for the signaling link selection (SLS) field of the MTP routing label of the MSU. This parameter is valid for SCCP protocol Class 1 messages only. For the TTC network variant, valid values are `0` through `15`. For all other variants excluding ANSI, the valid range is `0` through `31`. The SLS field determines the link over which the MSU is routed. You can route multiple MSUs over the same link by assigning the same SLS value to each MSU.

  In the ANSI network variant, the valid range of values is `0` through `31` if you specified a five-bit SLS (via selection of the default setting or selection through the `CHANGE-SLSTYPE` MML command. However, if you selected an eight-bit SLS using the `CHANGE-SLSTYPE` MML command, the valid range is `0-255`.

  NOTE

  If you set an eight-bit SLS in the `sccp_seq_control` field and a SINAP user specified use of a five-bit SLS (using the `CHANGE-SLS` MML command), the SINAP node masks out the upper three most significant bits. See “SINAP/SS7 Interaction with the SS7 Network” in Chapter 2 for more detailed information.

* `sccp_3rd_party_addr[MAX_ADDR_LEN]` (output)
  For a TCP/IP agent (registered at the SCCP boundary) receiving messages from a TCAP application, this field specifies the original calling party address information. The TCP/IP agent overwrites the original SCCP called party address information with its own point code and pseudo SSN to establish a two-way dialogue with an application registered at the TCAP boundary on the same SINAP node and system. In this case, the TCP/IP agent requires the original SCCP calling party address to correctly format and route messages back to the originating node over TCP/IP.

  For a TCAP application (accessed through the TCP/IP agent) originating a dialogue (for CCITT variants) or transaction (for ANSI variants), the field specifies the SCCP called party address of the TCAP application. In this case, the called party address is required because the original called party address provided in the `tblock` and `mblock` is

configured to address the own signaling point (OSP) code and pseudo SSN of the TCP/IP agent running on the same SINAP node.

The CASL transparently copies the `sccp_3rd_party_addr` field between the `tblock` and `mblock` in both directions when sending and receiving `tblocks`. The SINAP driver initializes this field in the `mblock` to zeros when the SINAP node receives messages from Level 2 of the SS7 network.

The constant specified in the `MAX_ADDR_LEN` parameter is defined in the include files `$SINAP_HOME/Include/mblock.h.` and `$SINAP_HOME/Include/tblock.h.`

* `importance_parm` (input/output)
  For the CCITT (ITU-T) variant, if the environment variable SCCP_ITU96_IMPORTANCE_PARM is set, and the user registers at the SCCPX boundary, this field holds the importance parameter (ss7-2392: 1996 ITU-T Q.713 3.19). The MSB is used as a bit flag to indicate if the SCCP optional Importance parameter is included in the SCCP XUDT/XUDTS message. If it is, then the 3 LSBs represent Importance values 0 to 7.

* `seg_included` (input)
  Flag to indicate if the Segmentation parameter was included in the SCCP XUDT message. Used by SINAP internally.

The MTP Control Structure (`mtp_ctrl_t`)

The `mtp_ctrl_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
    typedef struct  mtp_ctrl_s
    {
         U16      msg_id;
         U16      seq_number;
         U8       msg_type;
         U8       sender_id;
         U16      msg_size;
         mtp_ud_t user_data;
} mtp_ctrl_t;
```

The following fields are internal to the SINAP/SS7 system and you should not modify them:

* `msg_id` (output)
* `seq_number` (output)
* `msg_type` (output)
* `sender_id` (output)
* `user_data` (output)

The structure also contains this field for which you must specify values:

* `msg_size` (output)
  This field indicates the size of the MSU, including the length of the `bib_bsn`, `fib_fsn`, `li`, `sio`, and `label` fields. The field also indicates the length of MTP user data.

The MTP User Data Structure (`mtp_ud_t`)

The `mtp_ud_t` structure contains the following fields and is defined in the include file `mblock.h`.

NOTE

All of the structures within `mtp_ud_t` are defined in `mblock.h`.

```
typedef union
{
        U8              byte[8];
        U16             word[4];
        U8              ucomm;
        user_link_t     link;
        user_l2_t       to_l2;
        user_tcoc_t     tcoc;
        user_chg_t      chg;
        user_cong_t     cong;
        user_trsh_t     trsh;
} mtp_ud_t;
```

These fields are internal to the SINAP/SS7 system and you should not modify them.

* `byte` (output)
* `word` (output)
* `ucomm` (output)
* `link` (output) - See “The user_link_t Structure” later in this section.
* `to_l2` (output) - See “The user_l2_t Structure” later in this section.
* `tcoc` (output) - See “The user_tcoc_t Structure” later in this section.
* `chg` (output) - See “The user_chg_t Structure” later in this section.
* `cong` (output) - See “The user_cong_t Structure” later in this section.
* `trsh` (output) - See “The user_trsh_t Structure” later in this section.

The `user_link_t` Structure

The `user_link_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct  USR_LNK
{
        U8      link_set;
        U8      link_no;
} user_link_t;
```

These fields are internal to the SINAP/SS7 system and you should not modify them.

* `link_set` (output)
* `link_no` (output)

The `user_l2_t` Structure

The `user_l2_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct  USR_TL2
{
        U16     link_id;
        U8      byte[6];
} user_l2_t;
```

These fields are internal to the SINAP/SS7 system and you should not modify them.

* `link_id` (output)
* `byte[6]` (output)

The `user_tcoc_t` Structure

The `user_tcoc_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct  USR_TCOC
{
        U8      link_set;
        U8      link_no;
        U8      bsnt;
        U8      alt_link_set;
}user_tcoc_t;
```

These fields are internal to the SINAP/SS7 system and you should not modify them.

* `link_set` (output)
* `link_no` (output)
* `bsnt` (output)
* `alt_link_set` (output)

The `user_chg_t` Structure

The `user_chg_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct  USR_CHG
{
        U8      ref;
        U8      status;
} user_chg_t;
```

These fields are internal to the SINAP/SS7 system and you should not modify them.

* `ref` (output)
* `status` (output)

The `user_cong_t` Structure

The `user_cong_t` structure contains the following fields and is defined in the include file `mblock.h`.

> NOTE
>
> For important information about how the CCITT variant of the SINAP/SS7 system supports the national option of multiple link-congestion states without congestion priority, see the description of this field in the SINAP/SS7 `mblock.h` include file.

```
typedef struct  USR_CONG
{
        U8      link_set;
        U8      link_no;
        U8      status;
        U8      filler;
} user_cong_t;
```

These fields are internal to the SINAP/SS7 system and you should not modify them:

* `link_set` (output)
* `link_no` (output)
* `status` (output)
* `filler` (output)

The `user_trsh_t` Structure

The `user_trsh_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct  USR_TRSH
{
        U16     value;
        U8      level;
} user_trsh_t;
```

These fields are internal to the SINAP/SS7 system and you should not modify them:

* `value` (output)
* `level` (output)

The MSU Data Structure (`msu_t`)

The `msu_t` structure is the generic message structure that contains the following fields and is defined in the include file `mblock.h`. The following chart lists the versions of this message structure that are specific to each network variant:

Network Variant	Message Structure
CCITT	`ccitt_msu_t`
ANSI, China	`ansi_msu_t`
TTC, NTT	`ttc_msu_t`

Note that the China variant uses the ANSI message format.

When you develop an application, you can code the application so it uses the generic version of a CASL structure (for example, `msu_t`) or a particular variant-specific version (for example, `ttc_msu_t`). The global variable, `SINAP_VARIANT`, which defines the variant being used, links the generic versions of CASL structures to their corresponding variant-specific versions, thus making the source code consolidation invisible to programmers.

```
typedef struct msu_s
{
    U8      bib_bsn;
    U8      fib_fsn;
    U8      li;
    U8      sio;
    U32      label;           /* CCITT only - dpc-opc, sls/slc   */
    U8      dpc_member;       /* ANSI              */
    U8      dpc_cluster;      /* ANSI              */
    U8      dpc_network;      /* ANSI              */
    U8      opc_member;       /* ANSI              */
    U8      opc_cluster;      /* ANSI              */
    U8      opc_network;      /* ANSI              */
    U8      sls;            /* ANSI             */
    union mtp_ud_tag
    {
                U8   msg[SC_USER_MAX + 5];
                snm_user_t    snm;
                slt_user_t    slt;
                upu_user_t    upu;
                sccp_user_t   sccp;
                sccp_xuser_t sccpx; /* sccp data for XUDT */
        } mtp_ud;
 } msu_t;
```

* `bib_bsn` (output)
  This field is internal to the SINAP/SS7 system; you should not modify it.

* `fib_fsn` (output)
  This field is internal to the SINAP/SS7 system; you should not modify it.

* `li` (output)
  The two high-order bits (8 and 7) of this field specify the message priority of the MSU. This parameter is valid only for SCCP Class 0 and Class 1 messages. Valid values are in the range of 0 through 3 (lowest to highest). The remaining bits (6 through 1) specify the length of the `mtp_ud` field. This is the maximum usable data area in the `M_Block` (specified by `MAX_MBLK_DATA`).

* `sio` (output)
  Specifies the service information octet. This field is not set for this function, but for `ca_put_msu()`.

  The following fields are variant-specific. For the CCITT or TTC variant of the SINAP/SS7 system, use the `label` field. For the ANSI network variant, use the fields: `dpc_member`, `dpc_cluster`, `dpc_network`, `opc_member`, `opc_cluster`, `opc_network`, and `sls`.

  Format the fields according to the appropriate recommendations for the network variant of the SINAP/SS7 system being used. For CCITT, see the ITU-T (CCITT) Q.700 series of Recommendations. For ANSI, see the ANSI T1.111 Standards series.

* `label` (output)
  (CCITT) Specifies the DPC, OPC, and SLS of the MSU.

  (TTC) Defines this field as U8 label [6].

* `dpc_member` (output)
  (ANSI) Specifies the member-address component of the destination point code (DPC).

* `dpc_cluster` (output)
  (ANSI) Specifies the cluster-address component of the DPC.

* `dpc_network` (output)
  (ANSI) Specifies the network-address component of the DPC.

* `opc_member` (output)
  (ANSI) Specifies the member-address component of the originating point code (OPC).

* `opc_cluster` (output)
  (ANSI) Specifies the cluster-address component of the OPC.

* `opc_network` (output)
  (ANSI) Specifies the network-address component of the OPC.

* `sls` (output)
  (ANSI) Specifies the signaling link selection (SLS) field.

  NOTE

  You can use the macro `ANSI_CA_GET_SLS` to retrieve the actual SLS contained in this field. If a SINAP user specified a five-bit SLS (by using the default configuration or selecting a five-bit SLS using the `CHANGE-SLSTYPE` MML command) the value returned has the top three most significant bits zeroed (masked out). If a SINAP user specified an eight-bit SLS using the `CHANGE-SLSTYPE` MML command, the full eight-bits are returned. See "SINAP/SS7 Interaction with the SS7 Network" in Chapter 2 for more information.

These remaining fields apply to all variants of SINAP/SS7 (CCITT, ANSI, China, NTT, and TTC). Any differences are noted in the descriptions.

* `msg[SC_USER_MAX +5]` (output)
  Specifies the array for the `M_Block`. This field allows data access on an 8-bit boundary. (`SC_USER_MAX` is defined in the SINAP/SS7 `mblock.h` include file.)

* `snm` (output)
  Specifies an `snm_user_t` structure that contains the signaling network management data. For information about this structure, see "The Signaling Network Management Structure (snm_user_t)" later in this section.

  (TTC) The `ttc_snm_user_t` structure uses the structure, `ttc_snm_types_t`, in place of the fields, `snm_info_0` and `snm_info_1`. The `ttc_snm_user_t` and `ttc_snm_types_t` structures and their related structures (`ttc_snm_types_t`, `ttc_spec_info_t`, `ttc_snm_tfc_t`, and `ttc_snm_dpsc_t`) contain MTP management information. These structures are all internal to the SINAP/SS7 system and should not be modified.

* `slt` (output)
  Specifies an `slt_user_t` structure that contains the signaling link test data. For information about this structure, see "The Signaling Link Test Structure (slt_user_t)" later in this section.

  (TTC) Specifies the structure, `ttc_srt_user_t`, since the TTC variant performs signaling route testing (SRT) instead of signaling link testing (SLT).

* `sccp` (output)
  Specifies an `sccp_user_t` structure that contains the SCCP user data. For information about this structure, see "The SCCP User Data Structure (sccp_user_t)" later in this section.

(TTC) Defines a different length than the other variants for the `ttc_sccp_user_t` structure field, `ud[TTC_SC_USER_MAX]`.

* `upu` (output)
  Specifies the `upu_user_t` structure that contains the MTP user-part-unavailable (UPU) information.

  (TTC)Does not include the `upu_user_t` structure because the TTC variant does not support UPU messages.

The Signaling Network Management Structure (`snm_user_t`)

The `snm_user_t` structure contains the following fields and is defined in the include file `mblock.h`.

NOTE

The TTC variant uses the `ttc_snm_user_t` structure instead of the `snm_user_t` structure. The `ttc_snm_user_t` structure is also defined in the include file `mblock.h`.

```
typedef struct  snm_user_s
{
      U8         H0_H1;
      U8         snm_info_0;            /* CCITT only         */
      U8         snm_info_1;            /* CCITT only         */
      snm_types_t   snm_types;          /* ANSI only   */
} snm_user_t;
```

* `H0_H1` (output)
  This field is internal to the SINAP/SS7 system; you should not modify it.

The following fields are variant-specific. If you are using the CCITT or TTC network variants of the SINAP/SS7 system, use the fields, `snm_info_0` and `snm_info_1`. If you are using the ANSI variant, use the field `snm_types` instead.

* `snm_info_0` (output)
  (CCITT) This field is internal to the SINAP/SS7 system and you should not modify it.

  (TTC) Uses the structure, `ttc_snm_types_t`, which contains MTP management information.

* `snm_info_1` (output)
  (CCITT) This field is internal to the SINAP/SS7 system and you should not modify it.

(TTC) Uses the structure, `ttc_snm_types_t`, which contains MTP management information.

* `snm_types` (output)
  (ANSI) Specifies an `snm_types_t` structure that contains information about the MSU's signaling link code (SLC) and signaling point code (SPC), as well as transfer control (TFC) information for the MSU.

The Signaling Link Test Structure (`slt_user_t`)

The `slt_user_t` structure sends signaling link test (SLT) messages to determine whether a SINAP/SS7 link is operational. The structure contains the following fields and is defined in the include file `mblock.h`.

The TTC variant uses the structure, `ttc_srt_user_t`, since TTC performs signaling route test (SRT) messages instead of SLT messages.

```
typedef struct  slt_user_s
{
        U8      H0_H1;
        U8      li_spare;    /* CCITT - li_slc for ANSI  */
        U8      snm_signal_data[MAX_TEST_PATTERN];

} slt_user_t;
```

These fields are internal to the SINAP/SS7 system and you should not modify them.

* `H0_H1` (output)

* `snm_signal_data[MAX_TEST_PATTERN]` (output)
(`MAX_TEST_PATTERN` is defined in the SINAP/SS7 `mblock.h` include file.)

The following fields are variant-specific and internal to the SINAP/SS7 system. You should not modify them. Specify the field that is appropriate for your network variant.

* `li_spare` (output) (CCITT)
* `li_slc` (output) (ANSI)

The SCCP User Data Structure (`sccp_user_t`)

The `sccp_user_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct sccp_user_s
{
        U8      msg_type;
        U8      ret_prot;
        U8      cld_off;
        U8      clg_off;
        U8      tcap_off;
        U8      ud[SC_USER_MAX];
} sccp_user_t;
```

* `msg_type` (output)
  Specifies the message type for the SCCP message.

* `ret_prot` (output)
  Specifies the protocol class to use when sending the MSU and the return option which specifies the action to take if an error occurs. You use a single value to define both parameters, as shown in the following chart:

Value	Description
0	Connectionless Class 0, no return on error.
1	Connectionless Class 1, no return on error.
0 x 80	Connectionless Class 0, returns an error.
0 x 81	Connectionless Class 1, returns an error.

* `cld_off` (output)
  Specifies the offset to the called address for the SCCP message.

* `clg_off` (output)
  Specifies the offset to the calling address for the SCCP message.

* `tcap_off` (output)
  Specifies the offset to the data portion of the SCCP message.

* `ud[SC_USER_MAX]` (output)
  Specifies the SCCP message.

The `sccp_xuser_t` Structure

The `sccp_xuser_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct  sccp_xuser_s
{                                       /* partial M_BLOCK def  */
        U8      msg_type;
        U8      ret_prot;
        U8      hop_counter;
#define  HOPS           10      /* default hop count */
#define  MAX_HOPS       15      /* maximum hop count */

        U8      cld_off;                /* offset to called add */
        U8      clg_off;                /* offset to callng add */
        U8      tcap_off;               /* offset to TCAP data  */
        U8      op_off;                /* offset to optional data field */

                                           /*  for above fields    */
        U8      ud[CCITT_SC_USER_MAX - 2]; /* SCCP Header plus  */
                                           /* TCAP Header and data */
} sccp_xuser_t;
```

* `msg_type` (output)
  Specifies the message type for the SCCP message.

* `ret_prot` (output)
  Specifies the protocol class to use when sending the MSU and the return option which specifies the action to take if an error occurs. You use a single value to define both parameters, as shown in the following chart:

Value	Description
0	Connectionless Class 0, no return on error.
1	Connectionless Class 1, no return on error.
0 x 80	Connectionless Class 0, returns an error.
0 x 81	Connectionless Class 1, returns an error.

* `hop_counter(`output`)`
  The value of the hop counter is decremented on each global title translation and should be in range 15 to 1.

* `cld_off` (output)
  Specifies the offset to the called address for the SCCP message.

* `clg_off` (output)
  Specifies the offset to the calling address for the SCCP message.

* `tcap_off` (output)
  Specifies the offset to the data portion of the SCCP message.

* `ud[SC_USER_MAX]` (output)
  Specifies the SCCP message.

The `l3_event_t` Structure

The `l3_event_t` structure contains the following fields and is defined in the include file `event3.h`.

```
 typedef struct l3_event_s
{
     U32      dpc;
     U16      errnum;
     U32      label;
     U8       ttc_label[6];
     U8       sio;
     U8       link_set;
     U8       link_no;
     U8       ucomm;
} l3_event_t;
```

The following fields are internal to the SINAP/SS7 system and you should not modify them:

* `dpc` (output)
  Specifies the destination point code.

* `errnum` (output)

* `label` (input)
  Specifies the MTP routing label.

* `ttc_label[6]`
  (TTC) Specifies the routing label (`ttc_label[6]).`

* `sio` (input)

* `link_set` (input)

* `link_no` (input)

* `ucomm` (input)

The `iblk_t` Structure

The `iblk_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct iblk_s
{
        i_block_t iblk;                  /* iblock header */
        U8        data[MAX_MBLK_DATA - sizeof(i_block_t)];
/* MAX_MBLK_DATA=268 and sizeof(i_block_t)=248 in SINAP11.0 64-bit
*/
#ifdef _LP_32_64_
        U8      filler[4]; /* For User32/Driver64 compatibility */
#endif  /* _LP_32_64_ */
} iblk_t;
```

These fields are internal to the SINAP/SS7 system and you should not modify them:

* `iblk` (output) - See the include file `iblock.h` for a description of this structure.
* `data[MAX_MBLK_DATA]` (output) - (`MAX_MBLK_DATA` is defined in the `mblock.h` include file.)

FILES

`arch.h`, `ca_error.h`, `iblock.h`, `mblock.h`, SINAP/SS7`.h`, `sysdefs.h`, `sys/time.h`, `timestamp.h`, `sys/types.h`

RETURN VALUES

The `ca_get_msu()` function returns a pointer to the `M_Block`. If the function returns `-1`, there is an error; see `errno` for error number and description. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Possible UNIX values for `errno` are as follows.

Value	Meaning
`EBADF`	An invalid open file descriptor was specified.
`ENOTTY`	This `fides` is not associated with a device driver that accepts control functions.
`EFAULT`	The buffer points outside the process-allocated address space.
`EINVAL`	Queue ID is not a valid message queue ID. The value of `msg_type` is less than `1`, or `msg_sz` is less than `0` or greater than the system-imposed limit.

Value	Meaning
EIO	An I/O error occurred while reading or writing.
EINTR	The system call was interrupted by an UNIX signal.
ENODATA	There are no MSUs in the batch buffer.
ENXIO	The requested service cannot be performed on this particular subdevice.
ENOLINK	The link to a requested machine is no longer active.

Possible CASL values for `errno` are as follows.

Value	Meaning
CA_ERR_ACCESS	The process calling `ca_get_msu()` is not registered. Call `ca_register()` before calling this function.
CA_ERR_MSU_CALLS	The process is not registered at the MTP or SCCP boundary. Call `ca_register()` to reregister the process at one of these boundaries.
CA_ERR_NO_SS7_SVC	The process is not registered for SS7 service. Call `ca_register()` using `ss7_primitive=SS7_CTRL_PRIMITIVE` and `fss7=1` to reregister the process for SS7 service.

SEE ALSO

`ca_flush_msu()`, `ca_put_msu()`

ca_get_msu_noxudt()

SYNOPSIS

```
m_block_t         *ca_get_msu_noxudt(
                  BOOL              fwait);
```

DESCRIPTION

The `ca_get_msu_noxudt()` function is very similar to the `ca_get_msu()` function. The only difference is that the internal function `ca_get_msu_int()` is invoked with the `NOXUDT` option, so no `ca_xudt_reassemble` reassembly processing is performed.

See the `ca_get_msu()` function for a detailed description of structures referenced by `m_block_t`.

PARAMETERS

* `fwait`(input)
  Specifies whether the function is to wait for an MSU. Use 1 to indicate the function is to wait, otherwise, use 0. If you use 0, the function returns the error, `ENODATA`, when there are no MSUs.

FILES

`arch.h, ca_error.h, iblock.h, sinap.h, sysdefs.h, sys/time.h, timesstamp.h, sys/types.h`

RETURN VALUES

The `ca_get_msu_noxudt()` function returns a pointer to the `M_Block`. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning. See `sys/errno.h` for UNIX errors.

Possible UNIX values for `errno` are:

Value	Meaning
`EIO`	An input/output (I/O) error occurred while reading or writing.
`EFAULT`	The buffer points outside the process-allocated address space.

Value	Meaning
EINTR	The system call was interrupted by a UNIX signal.
ENODATA	There are no MSUs in the batch buffer.

Possible CASL values for `errno` are:

Value	Meaning
CA_ERR_ACCESS	The process calling `ca_get_msu()` is not registered. Call `ca_register()` before calling this function.
EFAULT	The process is not registered at the MTP or SCCP boundary. Call `ca_register()` to reregister the process at one of those boundaries.

SEE ALSO

`ca_flush_msu()`, `ca_get_msu()`, and `ca_put_msu()`.

ca_lookup_gt()

SYNOPSIS

```
#include <ca_gtt.h>

int ca_lookup_gt(gtt_tr_entry_t *tr_entry);
```

DESCRIPTION

The `ca_lookup_gt()` function performs a global-title lookup, returning the translation results for a particular global title. The function's only parameter, `*tr_entry`, is a pointer to a `gtt_tr_entry_t` structure.

When calling this function, you must initialize the appropriate `gtt_tr_entry_t` structure fields to define the global title whose entry you want to look up. If the specified global title matches a global-title entry, this function fills in the other structure fields with the replacement DPC, SSN, and/or address information for that global title. If the function does not find a match, it returns the error `GTT_ERR_NO_ENTRY` and leaves the other structure fields unchanged.

The `gtt_tr_entry_t` structure is defined in the `ca_gtt.h` include file and has the following format.

```
typedef struct gtt_tr_entry {
        struct gtt_tr_entry     *next;
#ifdef _LP_32_64_
        U32                     filler;  /* For User32/Driver64
compatibility */
#endif  /* _LP_32_64_ */
        struct gtt_tr_entry     *prev;
#ifdef _LP_32_64_
        U32                     filler1; /* For User32/Driver64
compatibility */
#endif  /* _LP_32_64_ */
        U8                      ssni;
        U8                      ssn;
        U8                      ssn2; /* ss7-1074.  Backup SSN. */
        U8                      pci;
        U32                     pc;
        U32                     dpc2;  /* ss7-1074.  Backup DPC. */
        U8                      state;
        U8                      gti;
        U8                      tt;
        U8                      np;
        U8                      es;
        U8                      noai;
        char                    laddr[MAX_GLOBAL_TITLE + 1];
        char                    haddr[MAX_GLOBAL_TITLE + 1];
        char                    naddr[MAX_GLOBAL_TITLE + 1];
#if defined(__LP64__) || defined(_LP_32_64_)
        U8                      filler2[7]; /* For User32/Driver64
compat.
*/
#endif  /* __LP64__ || _LP_32_64_ */
} gtt_tr_entry_t;
```

PARAMETERS

The following is a list of the `gtt_tr_entry_t` structure's fields. (An *input field* is a field that you must initialize, and an *output field* is one that the `ca_lookup_gt()` function fills in.)

* `*next`
  This field is internal to the SINAP/SS7 system; do not modify it.

* `*prev`
  This field is internal to the SINAP/SS7 system; do not modify it.

* `state` (output)
  This field is internal to the SINAP/SS7 system; do not modify it.

* `ssni` (output)
  Indicates whether an SSN is associated with the global title. If an SSN is associated with the global title, the value of this field is `1`; otherwise, it is `0`.

* `ssn` (output)
  The SSN associated with the global title.

* `ssn2` (output)
  The optional backup or secondary subsystem number to which GTT-related messages should be routed for processing if the primary subsystem number is unavailable.

* `pci` (output)
  Indicates whether a PC is associated with the global title. If a PC is associated with the global title, the value of this field is `1`; otherwise, it is `0`.

* `pc` (output)
  The PC associated with the global title.

* `dpc2` (output)
  The optional backup or secondary destination point code to which GTT-related messages should be routed for processing if the primary destination point code is unavailable.

* `gti` (input)
  The GTI for the global title. For CCITT and TTC applications, valid values are `1`, `2`, `3`, and `4`. For ANSI applications, valid values are `1` and `2`.

* `tt` (input)
  The translation type for the global title. Valid values are in the range `0` to `254`. CCITT and TTC applications must specify a value for this field if the `gti` field is `2`, `3`, or `4`. ANSI applications must specify a value for this field regardless of the value of the `gti` field.

* `np` (input)
  The numbering plan for the global title. Valid values are in the range `0` to `14`. CCITT and TTC applications must specify a value for this field if the `gti` field is `2` or `3`. ANSI applications must specify a value for this field if the `gti` field is `1`.

* `es` (output)
  The encoding scheme for the global title. The encoding scheme is part of the numbering plan and currently is not implemented.

* `noai` (input)
  The nature-of-address indicator for the global title. Valid values are in the range of `1` through `127` if the environment variable `GTT_BYPASS_NOAI_CHECK` is defined. If the environment variable is not defined, the range of values allowed is `1` through `4`. CCITT and TTC applications must specify a value for this field if the `gti` field is `1` or `4`. ANSI applications should not specify a value for this field.

* `laddr[MAX_GLOBAL_TITLE + 1]` (input)
  The low-address information for the global title.

* `haddr[MAX_GLOBAL_TITLE + 1]` (output)
  The high address of a range of global titles. This field is optional.

* `naddr[MAX_GLOBAL_TITLE + 1]` (output)
  The new address information to substitute for the original address information in the global title.

FILES

`ca_gtt.h`

RETURN VALUES

The value `RET_OK` indicates that a matching global-title entry was found.

If an error occurs the function returns `-1` and *`errno`* is set to one of the following error codes.

NOTE

When `ca_lookup_gt()` returns an error, the values in the `gtt_tr_entry_t` structure fields remain unchanged.

Error Code	Numeric Value	Description
`GTT_ERR_BAD_GTI`	1	The specified GTI value is invalid.
`GTT_ERR_BAD_TT`	2	The specified translation type is invalid.
`GTT_ERR_BAD_NP`	3	The specified numbering plan is invalid.
`GTT_ERR_BAD_NOAI`	5	The specified nature-of-address indicator is invalid.
`GTT_ERR_BAD_LADDR`	9	The specified low-address information is invalid.
`GTT_BAD_ERR_FAULT`	14	The pointer to the `gtt_tr_entry_t` structure is invalid.
`GTT_ERR_NO_ENTRY`	16	None of the global-title entries match the specified global title.

ca_put_msu()

SYNOPSIS

```
int     ca_put_msu(
          m_block_t    *pmblk);
```

DESCRIPTION

The `ca_put_msu()` function sends an `M_Block` containing an MSU. The `ca_put_msu()` function accumulates MSUs in an output batch buffer and passes a batch of messages to the SS7 driver when the buffer becomes full. The client batch size is declared at registration (see the description of `ca_register()` for more information).

Only those applications registered for MTP or SCCP services should call this function. If the application is registered to receive input at the SCCP boundary, this function sends the MSU to the SCCP. If the application is registered to receive input at the MTP boundary, the function sends the MSU directly to the MTP. Applications registered for TCAP services should **not** call this function; instead, they should call `ca_put_tc()` to send a `T_Block`.

PARAMETERS

* `pmblk` (input)

  Specifies a pointer to the `M_Block` that contains the MSU being sent. The `M_Block` is defined in the `m_block_t` structure, which is defined in the include file `mblock.h`.

  The fields in the `m_block_t` structure must specify either SCCP or MTP routing.

For SCCP routing, you must specify values for the following fields in the `m_block_t` structure.

```
ts
sc_ctrl.sccp_ctrl (should be set to N-UNITDATA)
ud.msu.mtp_ud.sccp.msg_type
ud.msu.mtp_ud.sccp.ret_prot
ud.msu.mtp_ud.sccp.cld_off
ud.msu.mtp_ud.sccp.clg_off
ud.msu.mtp_ud.sccp.tcap_off
ud.msu.mtp_ud.sccp.ud[SC_USER_MAX]
mtp_ctrl.msg_size
```

For MTP routing, you must specify values for the following fields in the `m_block_t` structure.

```
ts
ud.msu.sio
ud.msu.label
ud.msu.mtp_ud.msg[MAX_MBLK_DATA]
mtp_ctrl.msg_size
```

You should specify `0` for any unused fields. For an explanation of these fields and possible values, see the following section, “The Main M_Block Structure (`m_block_t`).”

The Main M_Block Structure (m_block_t)

The m_block_t structure contains the following fields and is defined in the include file mblock.h.

```
 typedef  struct  m_block_s
{
        ca_ctrl_t         ca_ctrl;
        timestamp_t       ts;
        bi_ctrl_t         bi_ctrl;
        tcap_ctrl_t       tc_ctrl;
        sccp_ctrl_t       sc_ctrl;
        sccp_prim_t       sc_prim;
        tcap_alt_t        tc_alt;
        mtp_ctrl_t        mtp_ctrl;
        union m_block_ud_tag
        {
          msu_t           msu;
          ccitt_msu_t     ccitt_msu; /*CCITT MSU Data */
          ttc_msu_t       ttc_msu;   /*TTC and NTT MSU Data */
          ansi_msu_t      ansi_msu;  /*ANSI and China MSU Data */
       13_event_t      dr_event;
         iblk_t          ib;
        } ud;
} m_block_t;
```

* ca_ctrl (output)
  Specifies CASL control information. For information about this structure's fields, see "The CASL Control Structure (ca_ctrl_t)" later in this section.

* ts (output)
  Specifies a collection of timestamps that the SINAP/SS7 system automatically inserts. The timestamps aid monitoring and logging, are visible when you run the BITE log-analysis program. For information about this structure's fields, see "The Timestamp Structure (timestamp_t)" later in this section.

* bi_ctrl (output)
  Specifies BITE control information. For information about this structure's fields, see "The BITE Control Structure (bi_ctrl_t)" later in this section.

* tc_ctrl (output)
  Specifies TCAP control information. For information about this structure's fields, see "The TCAP Control Structure (tcap_ctrl_t)" later in this section.

* `sc_ctrl` (output)
  Specifies SCCP control information. For information about this structure's fields, see "The SCCP Control Structure (`sccp_ctrl_t`)" later in this section.

* `sc_prim` (input)
  Specifies the `sccp_prim_t` structure which conveys information about large messages. This structure is defined in the `Mblock.h` include file. See "The `sccp_prim_t` Structure" later in this chapter for more information.

* `tc_alt` (output)
  The SINAP/SS7 system uses this field for specifying the alternative DPC (refer to chapter 3) to be filled at the MTP Routing Label's DPC field of the outbound MSU. For information about this structure's fields, see "The TCAP Alternative DPC Structure (`tcap_alt_t`)" later in this section.

* `mtp_ctrl` (output)
  Specifies MTP control information. For information about this structure's fields, see "The MTP Control Structure (`mtp_ctrl_t`)" later in this section.

* `ud` (output)
  Specifies the union of `M_Block`.

  - `msu` (output)
    Specifies user data for the MSU or `I_Block` information. See "The MSU Data Structure (`mtp_ud_t`)" later in this section for information about this structure's fields.
  - `dr_event` (output)
    This field is internal to the SINAP/SS7 system; you should not modify it. For information about this structure's fields, see "The `l3_event_t` Structure" later in this section.
  - `ib` (output)
    See "The `i_blk_t` Structure" later in this section for information about this structure's fields.
  - `ccitt_msu` (output)
    Specifies MSU data for the CCITT network variant.
  - `ttc_msu` (output)
    Specifies MSU data for the TTC and NTT network variants.
  - `ansi_msu` (output)
    Specifies MSU data for the ANSI and China network variants.

The `l3_event_t` Structure

The `l3_event_t` structure contains the following internal fields and is defined in the include file `event3.h`.

```
 typedef struct l3_event_s
{
     U32      dpc;
     U16      errnum;
     U32      label;
     U8       ttc_label[6];
     U8       sio;
     U8       link_set;
     U8       link_no;
     U8       ucomm;
} l3_event_t;
```

The following fields are internal to the SINAP/SS7 system and you should not modify them:

* `dpc` (input)
* `errnum` (output)
* `label` (input)
  Specifies the MTP routing label.
* `ttc_label[6]`
  (TTC) Specifies the routing label (`ttc_label[6]`).
* `sio` (input)
* `link_set` (input)
* `link_no` (input)
* `ucomm` (input)

The CASL Control Structure (`ca_ctrl_t`)

The following fields make up the `ca_ctrl_t` structure, which is defined in the include file `blkhdr.h`.

```
typedef struct   ca_ctrl_s
{
       int     msg_type;       /* For compatibility with the existing UNIX IPC
                                     mechanism. */
       int     msu_cnt;        /* # of MSUs pending */
       int     free_cnt;       /* # of free MSUs in read queue */
       int     wfree_cnt;      /* # of free MSUs in write queue */
       S16     lost_cnt;       /* # of MSUs lost due to insuff. resources */
       S16     data_size;      /* Total size of structure excluding this
                                  structure. */
       U8      node_index;     /* index (0 - 3) of current node */
       U8      sinap_variant;  /* V_CCITT, V_ANSI, V_HYBRID, V_TTC */
       U16     link;           /* index of the origination link */
       pid_t   pid;            /* Process ID of a specific process or 0
                                  for load distribution */
       int     msg_sender;     /* Set to 0 if from link otherwise contains
                                  the process ID */
       U8      iblk;           /* TRUE if data contains I_Block */
                               /* Not used anywhere!!!!! */
       U8      rw;             /* Flag for monitor message only */
       U8      monitor_id;     /* monitor ID for monitored MSU  */
       U8      ssn_sio;        /* SSN or SIO ID for load distribution. */
       struct {
               U32     timer_id;       /* For CASL internal use only */
               U32     timer_val;      /* For CASL internal use only */
               int     omsg_type;      /* For CASL internal use only */
       } timr;
       struct {                        /* For internal use only */
               int     source;         /* For distribution managment. */
               int     destination;    /* For protocol processing. */
#ifdef _KERNEL
               mblk_t  *mptr;          /* Back pointer to mblk_t.  L3 only.*/
#else
               void    *mptr;          /* ss7-1102 - dummy back pointer. */
#ifdef _LP_32_64_
               U32  filler;            /* For User32/Driver64 compatibility*/
#endif  /* _LP_32_64_ */
#endif  /* _KERNEL */
       } internal;
} ca_ctrl_t;
```

* `msg_type` (output)
  Specifies the type of MSU being sent. This field is compatible with the existing UNIX IPC mechanism.

* `data_size` (output)
  Specifies the total size of the structure, excluding this field.

* `node_index` (output)
  This is an internal field that is automatically initialized to the appropriate value for the SINAP node.

* `sinap_variant` (output)
  This is an internal field that is automatically initialized to the appropriate value for the network variant being used on the SINAP node.

* `lost_cnt` (output)
  Specifies the number of `M_Blocks` lost due to insufficient resources within the SS7 driver.

* `msu_cnt` (output)
  Specifies the number of MSUs pending.

* `free_cnt` (output)
  Specifies the number of free MSUs pending in the read queue.

* `wfree_cnt` (output)
  Specifies the number of free MSUs pending in the write queue.

The following fields are internal to the SINAP/SS7 system and you should not modify them:

* `pid` (output)
* `link` (output)
* `msg_sender` (output)
* `iblk` (output)
* `rw` (output)
* `monitor_id` (output)
* `ssn_sio` (output)
* `source` (output)
* `destination` (output)
* `mptr` (input) Specifies a pointer to `m_block_t`, Level 3.
* `timer_id` (output)
* `timer_val` (output)
* `omsg_type` (output)

The Timestamp Structure (`timestamp_t`)

The `timestamp_t` structure contains the following fields and is defined in the include file `timestamp.h`. .

```
typedef struct timestamp_s
{
         U16     index;
         stamp_t stamp[MAX_TIME_STAMPS];
}timestamp_t;
```

* `index` (input)
  Specifies the next slot to stamp the time.

* `stamp[MAX_TIME_STAMPS]` (input)
  Specifies the timestamp slots. For an explanation, see "The `stamp_t` Structure," which follows. (`MAX_TIME_STAMPS` is defined in the SINAP/SS7 `timestamp.h` include file.)

The `stamp_t` Structure

The `stamp_t` structure contains the following fields and is defined in the include file `timestamp.h`.

```
typedef struct stamp_s
{
       U32     secs;                   /* time in seconds since 1/1/70 */
       U8      tsid;                   /* timestamp id                 */
       U8      ipcx;                   /* ipc index if applicable      */
       U16     msec;                   /* time in milliseconds         */
} stamp_t;
```

* `secs` (input)
  Specifies the time (in seconds) since 1/1/70.

* `tsid` (input)
  Specifies the timestamp ID. Valid values are defined in the include file `timestamp.h`.

* `ipcx` (input)
  Specifies the IPC index, if applicable.

* `msec` (input)
  Specifies the time, in milliseconds.

The BITE Control Structure (`bi_ctrl_t`)

The `bi_ctrl_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct  bi_ctrl_s
{
       U8      command;                /* Command type */
       U8      qualifier;              /* command qualifier*/
       U8      rw;                     /* monitor read/write/both */
       U8      monitor_id;
       U8      filler[2];              /* For User32/Driver64 compatibility*/
       U16     link;                   /* link index */
       pid_t   pid[2];                 /* application and SE process IDs*/
} bi_ctrl_t;
```

The following fields are internal to the SINAP/SS7 system and you should not modify them:

* `command` (input)
* `qualifier` (input)
* `link` (input) - The UNIX include file `sys/types.h` defines the `dev_t` structure.
* `pid[2]` (input)
* `rw` (input)
* `monitor_id` (input)

The TCAP Control Structure (`tcap_ctrl_t`)

The `tcap_ctrl_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct    tcap_ctrl_s
{
     S16    ipc_index;
     S32    trans_id;
     U8     tcap_msg_type;
     U8     abort_type;
     U8     abort_cause;
     U8     call_disposition;
} tcap_ctrl_t;
```

The following fields are internal to the SINAP/SS7 system and you should not modify them:

* `ipc_index` (input)

* `trans_id` (input)

* `tcap_msg_type` (input)

* `abort_type` (input)

* `abort_cause` (input)

* `call_disposition` (input)
  (ANSI) Specifies that the FOPC will be used in the MTP header. Used in the ANSI network variant only.

The SCCP Control Structure (`sccp_ctrl_t`)

The `sccp_ctrl_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct    sccp_ctrl_s
{
       U8     sccp_ctrl;
       U8     sccp_source;
       U8     sccp_dest;
       U8     ccitt_sls5;
       U16    sccp_err_no;
       U8     sccp_msg_priority;
       U8     sccp_seq_control;
       U8     sccp_3rd_party_addr[MAX_ADDR_LEN];
       U8     importance_parm;
       U8     seg_included;
       U8     filler1;
       U8     filler2;
} sccp_ctrl_t;
```

* `sccp_ctrl` (input)
  This field is meaningful to the SCCP user. SCCP Management sets this field to `N_UNITDATA` or `N_NOTICE`.

* `sccp_source` (input)
  This field specifies which SCCP function originated the message; you should not modify it.

* `sccp_dest` (input)
  This field specifies the SCCP function for which the message is destined; you should not modify it.

* `ccitt_sls5` (output)
  (CCITT) This is a five-bit SLS field used only in the CCITT network variant. Bit 4 of the five-bit SLS is used to randomly choose the route/link set (0 or 1) *if* the route set (RSET) is configured with two route/link sets for load sharing. The four-bit SLS value of the MSU label maps to a corresponding physical link in that route/link set used to send the outbound message. This field is used only to send outbound MSUs for MTP, SCCP, or TCAP applications. The field is not used to determine the route/link set that is used for inbound SS7 messages or the outbound SLT or SNM messages. You should not modify this field.

* `sccp_err_no` (input)
  This field indicates the SCCP error; you should not modify it.

* `sccp_msg_priority` (input)
  This field specifies the message priority for the MSU. This priority parameter is valid only for SCCP Class 0 and Class 1 messages. Valid values are `0` through `3` (lowest to highest priority, respectively).

* `sccp_seq_control` (input)
  This field specifies the value to use for the signaling link selection (SLS) field of the MSU's MTP routing label. This parameter is valid for SCCP protocol Class 1 messages only. Valid values for the TTC network variant are 0 through 15. For all other variants excluding ANSI, the valid range is `0` through `31`. The SLS field determines the link over which the MSU is routed. You can route multiple MSUs over the same link by assigning the same SLS value to each MSU.

  In the ANSI network variant, the valid range of values is `0` through `31` if you specified a five-bit SLS (via selection of the default setting or selection through the `CHANGE-SLSTYPE` MML command. However, if you selected an eight-bit SLS using the `CHANGE-SLSTYPE` MML command, the valid range is `0-255`.

  NOTE

  If you set an eight-bit SLS in the SCCP_seq_control field and a SINAP user specified use of a five-bit SLS (using the `CHANGE-SLS` MML command), the SINAP node masks out the upper three most significant bits. See "SINAP/SS7 Interaction with the SS7 Network" in Chapter 2 for more detailed information.

* `sccp_3rd_party_addr[MAX_ADDR_LEN]` (input)
  For a TCP/IP agent (registered at the SCCP boundary) receiving messages from a TCAP application, this field specifies the original calling party address information. The TCP/IP agent overwrites the original SCCP called party address information with its own point code and pseudo SSN to establish a two-way dialogue with an application registered at the TCAP boundary on the same SINAP node and system. In this case, the TCP/IP agent requires the original SCCP calling party address to correctly format and route messages back to the originating node over TCP/IP.

  For a TCAP application (accessed through the TCP/IP agent) originating a dialogue (for CCITT variants) or transaction (for ANSI variants), the field specifies the SCCP called party address of the TCAP application. In this case, the called party address is required because the original called party address provided in the tblock and mblock is configured to address the own signaling point (OSP) code and pseudo SSN of the TCP/IP agent running on the same SINAP node.

  The CASL transparently copies the `sccp_3rd_party_addr` field between the `tblock` and `mblock` in both directions when sending and receiving `tblocks`. The SINAP driver initializes this field in the `mblock` to zeros when the SINAP node receives messages from Level 2 of the SS7 network.

  The constant specified in the `MAX_ADDR_LEN` parameter is defined in the include files `$SINAP_HOME/Include/mblock.h.` and `$SINAP_HOME/Include/tblock.h.`

* `importance_parm` (input/output)
  For the CCITT (ITU-T) variant, if the environment variable SCCP_ITU96_IMPORTANCE_PARM is set, and the user registers at the SCCPX boundary, this field holds the importance parameter (ss7-2392: 1996 ITU-T Q.713 3.19). The MSB is used as a bit flag to indicate if the SCCP optional Importance parameter is included in the SCCP XUDT/XUDTS message. If it is, then the 3 LSBs represent Importance values 0 to 7.

* `seg_included` (input)
  Flag to indicate if the Segmentation parameter was included in the SCCP XUDT message. Used by SINAP internally.

The TCAP Alternative DPC Structure (`tcap_alt_t`)

The `tcap_alt_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef union
{
#if defined(__LP64__) || defined(_LP_32_64_)
        U64 filler; /* For User32/Driver64 compatibility */
#endif  /* __LP64__ || _LP_32_64_ */
       alt_dpc_t alt_dpc; /* For ANSI & China variants */
       alt_ccitt_t alt_ccitt; /* For CCITT variant */
} tcap_alt_t;

typedef struct
{
        U8      member; /* ANSI alternative DPC's member */
        U8      cluster;/* ANSI alternative DPC's cluster */
        U8      network;/* ANSI alternative DPC's network */
        U8      status; /* set to 1 to use alternative DPC */
} alt_dpc_t;

typedef struct
{
        U16     dpc; /* CCITT alternative DPC */
        U8      status; /* set to 1 to use alternative DPC */
        U8      filler; /* for alignment purpose */
} alt_ccitt_t;
```

* `alt_dpc` (input)
  For a SINAP ANSI or China TCAP user application that uses the alternative DPC feature (see chapter 3), the fields of the `m_block_t tc_alt.alt_dpc` data structure, i.e. ANSI or China DPC (member, cluster and network) and status, are copied internally by SINAP from `t_block_t thp.alt_DPC[DPC_LEN]` and `tb_options` respectively. See "The `tc_thp_t` Structure" under `ca_put_tc()` later in this chapter

for more information. For SINAP ANSI or China MTP and SCCP user applications, however, these fields of `m_block_t tc_alt.alt_dpc` data structure are set directly by the SINAP MTP or SCCP user application in order to use the alternative DPC feature before invoking the `ca_put_msu()` API.

* `alt_ccitt` (input)
  For SINAP CCITT TCAP user application using Alternative DPC feature (see chapter 3), the two fields of m_block_t tc_alt.alt_ccitt data structure, i.e. CCITT dpc and status, are copied internally by SINAP from t_block_t dhp.alt_DPC and tb_options respectively. See "The `tc_dhp_t` Structure" under ca_put_tc() later in this chapter for more information. However, SINAP CCITT MTP or SCCP user application set these fields of m_block_t tc_alt.alt_ccitt data structure directly in order to use Alternative DPC feature before invoking ca_put_msu() API.

The MTP Control Structure (`mtp_ctrl_t`)

The `mtp_ctrl_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
   typedef struct  mtp_ctrl_s
   {
        U16      msg_id;
        U16      seq_number;
        U8       msg_type;
        U8       sender_id;
        U16      msg_size;
        mtp_ud_t user_data;
} mtp_ctrl_t;
```

The following fields are internal to the SINAP/SS7 system and you should not modify them:

* `msg_id` (input)
* `seq_number` (input)
* `msg_type` (input)
* `sender_id` (input)
* `user_data` (input)

You must specify information for the following field:

* `msg_size` (input)
  This field indicates the size of the MSU, including the length of the `bib_bsn`, `fib_fsn`, `li`, `sio`, and `label` fields. The field also indicates the length of MTP user data.

The MTP User Data Structure (`mtp_ud_t`)

The `mtp_ud_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef union
{
        U8              byte[8];
        U16             word[4];
        U8              ucomm;
        user_link_t     link;
        user_l2_t       to_l2;
        user_tcoc_t     tcoc;
        user_chg_t      chg;
        user_cong_t     cong;
        user_trsh_t     trsh;
} mtp_ud_t;
```

The following fields are internal to the SINAP/SS7 system and you should not modify them:

* `byte` (input)
* `word` (input)
* `ucomm` (input)
* `link` (input) - See “The `user_link_t` Structure” later in this section.
* `to_l2` (input) - See “The `user_l2_t` Structure” later in this section.
* `tcoc` (input) - See “The `user_tcoc_t` Structure” later in this section.
* `chg` (input) - See “The `user_chg_t` Structure” later in this section.
* `cong` (input) - See “The `user_cong_t` Structure” later in this section.
* `trsh` (input) - See “The `user_trsh_t` Structure” later in this section.

The `user_link_t` Structure

The `user_link_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct  USR_LNK
{
        U8      link_set;
        U8      link_no;
} user_link_t;
```

The following fields are internal to the SINAP/SS7 system and you should not modify them:

* `link_set` (input)
* `link_no` (input)

The `user_l2_t` Structure

The `user_l2_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct  USR_TL2
{
        U16     link_id;
        U8      byte[6];
} user_l2_t;
```

The following fields are internal to the SINAP/SS7 system and you should not modify them:

* `link_id` (input)
* `byte[6]` (input)

The `user_tcoc_t` Structure

The `user_tcoc_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct  USR_TCOC
{
        U8      link_set;
        U8      link_no;
        U8      bsnt;
        U8      alt_link_set;
}user_tcoc_t;
```

The following fields are internal to the SINAP/SS7 system and you should not modify them:

* `link_set` (input)
* `link_no` (input)
* `bsnt` (input)
* `alt_link_set` (input)

The `user_chg_t` Structure

The `user_chg_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct  USR_CHG
{
        U8      ref;
        U8      status;
} user_chg_t;
```

The following fields are internal to the SINAP/SS7 system and you should not modify them:

* `ref` (input)
* `status` (input)

The `user_cong_t` Structure

The `user_cong_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct  USR_CONG
{
        U8      link_set;
        U8      link_no;
        U8      status;
        U8      filler;
} user_cong_t;
```

The following fields are internal to the SINAP/SS7 system and you should not modify them:

* `link_set` (input)
* `link_no` (input)
* `status` (input)
* `filler` (input)

The `user_trsh_t` Structure

The `user_trsh_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct  USR_TRSH
{
        U16     value;
        U8      level;
} user_trsh_t;
```

The following fields are internal to the SINAP/SS7 system and you should not modify them:

* `value` (input)
* `level` (input)

The MSU Data Structure (`msu_t`)

The `msu_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct msu_s
{
        U8      bib_bsn;
        U8      fib_fsn;
        U8      li;
        U8      sio;
        U32     label;          /* CCITT only - dpc-opc, sls/slc   */
        U8      dpc_member;     /* ANSI             */
        U8      dpc_cluster;    /* ANSI             */
        U8      dpc_network;    /* ANSI             */
        U8      opc_member;     /* ANSI            */
        U8      opc_cluster;    /* ANSI             */
        U8      opc_network;    /* ANSI             */
        U8      sls;            /* ANSI             */
        union mtp_ud_tag
        {
                U8   msg[SC_USER_MAX + 5];
                snm_user_t    snm;
                slt_user_t    slt;
                upu_user_t    upu;
                sccp_user_t  sccp;
                sccp_xuser_t sccpx; /* sccp data for XUDT */
        } mtp_ud;
 } msu_t;
```

* `bib_bsn` (input)
  This field is internal to the SINAP/SS7 system; you should not modify it.

* `fib_fsn` (input)
  This field is internal to the SINAP/SS7 system; you should not modify it.

* `li` (input)
  The two high-order bits(8 and 7) specify the message priority of the MSU. This priority parameter is valid only for SCCP Class 0 and Class 1 messages. The valid range for priority is 0 (lowest) through 3 (highest). The remaining bits (6 through 1) specify the length of the `mtp_ud` field. This is the maximum usable data area in the `M_Block`, which is specified by `MAX_MBLK_DATA`. (`MAX_MBLK_DATA` is defined in the SINAP/SS7 `mblock.h` include file.)

* `sio` (input)
  Specifies the service information octet.

  The following fields are variant-specific. If you are using the CCITT, NTT, or TTC network variant of the SINAP/SS7 system, use the `label` field. If you are using the ANSI or China network variant, use the fields: `dpc_member`, `dpc_cluster`, `dpc_network`, `opc_member`, `opc_cluster`, `opc_network`, and `sls`.

  Code these fields according to the Recommendations for the type of application you are developing. For CCITT applications, see the ITU-T (CCITT) Q.700 series of Recommendations. For ANSI applications, see the ANSI T1.111 Standards.

* `label` (input)
  (CCITT) Specifies the destination point code (DPC), originating point code (OPC), and signaling link selection (SLS) of the MSU.

  (TTC) Defines this field as U8 label [5].

* `dpc_member` (input)
  (ANSI, China) Specifies the member address component of the DPC.

* `dpc_cluster` (input)
  (ANSI, China) Specifies the cluster address component of the DPC.

* `dpc_network` (input)
  (ANSI, China) Specifies the network address component of the DPC.

* `opc_member` (input)
  (ANSI, China) Specifies the member address component of the OPC.

* `opc_cluster` (input)
  (ANSI, China) Specifies the cluster address component of the OPC.

* `opc_network` (input)
  (ANSI, China) Specifies the network address component of the OPC.

* `sls` (input)
  (ANSI, China) Specifies the SLS field.

> NOTE
>
> You can use the macro `ANSI_CA_SET_LABEL` to set either five-bit or eight-bit SLS in the MTP routing label. See "Setting SLS Bits in the MTP Routing Label" in Chapter 3 for more information.

The following fields apply to all variants. Any differences are noted in the descriptions:

* `msg[SC_USER_MAX +5]` (input)
  Specifies the array for the `M_Block`; `SC_USER_MAX +5` specifies the size of the array. This field allows data access on an 8-bit boundary. (`SC_USER_MAX` is defined in the SINAP/SS7 `mblock.h` include file.)

* `snm` (input)
  Specifies an `snm_user_t` structure that contains the signaling network management data. For information about this structure, see "The Signaling Network Management Data Structure (`snm_user_t`)" later in this section.

  (TTC and NTT) Specifies the `ttc_snm_user_t` structure.

* `slt` (input)
  Specifies an `slt_user_t` structure that contains the signaling link test data. For information about this structure, see "The Signaling Link Test Structure (`slt_user_t`)" later in this section.

  (TTC and NTT) Specifies the `ttc_srt_user_t` structure since TTC performs signaling route tests (SRTs) instead of signaling link tests (SLTs).

* `upu` (output)
  Specifies the `upu_user_t` structure that contains the MTP user-part-unavailable (UPU) information.

  (TTC and NTT) Does not include the `upu_user_t` structure because the TTC and NTT variants do not support user part unavailable (UPU) messages.

* `sccp` (input)
  Specifies an `sccp_user_t` structure that contains the SCCP user data. For information about this structure, see "The SCCP User Data Structure (`sccp_user_t`)" later in this section.

(TTC and NTT) Defines a different length than the other variants for the `ttc_sccp_user_t` structure field, `ud[TTC_SC_USER_MAX]`.

The Signaling Network Management Structure (`snm_user_t`)

The `snm_user_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct  snm_user_s
{
        U8         H0_H1;
        U8         snm_info_0;         /* CCITT, NTT, and TTC */
        U8         snm_info_1;         /* CCITT, NTT, and TTC */
        snm_types_t  snm_types;        /* ANSI/only      */

} snm_user_t;
```

* `H0_H1` (input)
  This field is internal to the SINAP/SS7 system and you should not modify it.

  The following fields are variant-specific. If you are using the CCITT, NTT, or TTC network variants of the SINAP/SS7 system, use the fields, `snm_info_0` and `snm_info_1`. If you are using the ANSI variant, use the field, `snm_types`, instead.

* `snm_info_0` (output)
  (CCITT) This field is internal to the SINAP/SS7 system and you should not modify it.

  (TTC) Uses the structure, `ttc_snm_types_t`, which contains MTP management information.

* `snm_info_1` (output)
  (CCITT) This field is internal to the SINAP/SS7 system and you should not modify it.

  (TTC) Uses the structure, `ttc_snm_types_t`, which contains MTP management information.

* `snm_types` (output)
  (ANSI) Specifies an `snm_types_t` structure that contains information about the MSU's signaling link code (SLC) and signaling point code (SPC), as well as transfer control (TFC) information for the MSU.

The Signaling Link Test Structure (`slt_user_t`)

The `slt_user_t` structure is used for sending signaling link test (SLT) messages to determine whether a SINAP/SS7 link is operational. The structure contains the following fields and is defined in the include file `mblock.h`.

The TTC variant uses the `ttc_srt_user_t` structure since TTC performs signaling route testing (SRT) instead of SLT.

```
typedef struct  slt_user_s
{
        U8      H0_H1;
        U8      li_spare;    /* CCITT - li_slc for ANSI*/
        U8      snm_signal_data[MAX_TEST_PATTERN];

} slt_user_t;
```

* `H0_H1` (input)
  Specifies the header code for the signaling link test control (SLTC) message. The lower four bits specify the message group (H0) and the upper four bits specify the signal code (H1).

* `snm_signal_data[MAX_TEST_PATTERN]` (input)
  Specifies the data you want to include in the signaling link test (SLT) message. (`MAX_TEST_PATTERN` is defined in the `mblock.h` include file.)

The following fields are specific to the network variant you are using. Use the one appropriate for your variant and specify the required information:

* `li_spare` (input)
  (CCITT) Specifies the length of the data in the test message (upper four bits); the lower four bits are spare.

* `li_slc` (input)
  (ANSI) Specifies the length of the data in the test message (upper four bits) and the SLC code of the link being tested (lower four bits).

The SCCP User Data Structure (sccp_user_t)

The sccp_user_t structure contains the following fields and is defined in the include file mblock.h.

```
typedef struct  sccp_user_s
{
        U8      msg_type;
        U8      ret_prot;
        U8      cld_off;
        U8      clg_off;
        U8      tcap_off;
        U8      ud[SC_USER_MAX];
} sccp_user_t;
```

* msg_type (input)
  Specifies the message type for the SCCP message.

* ret_prot (input)
  Specifies the return protection for the SCCP message. Valid values are in the range 0 through 3.

* cld_off (input)
  Specifies the offset to the called address for the SCCP message.

* clg_off (input)
  Specifies the offset to the calling address for the SCCP message.

* tcap_off (input)
  Specifies the offset to the data portion of the SCCP message.

* ud[SC_USER_MAX] (input)
  Specifies the SCCP message. (SC_USER_MAX is defined in the SINAP/SS7 mblock.h include file.)

  (TTC) Defines a different length than the other variants for the ttc_sccp_user_t structure field, ud[TTC_SC_USER_MAX] in the mblock.h include file.

The `sccp_xuser_t` Structure

The `sccp_xuser_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct  sccp_xuser_s
{                                       /* partial M_BLOCK def  */
        U8      msg_type;
        U8      ret_prot;
        U8      hop_counter;
#define  HOPS           10      /* default hop count */
#define  MAX_HOPS       15      /* maximum hop count */

        U8      cld_off;                /* offset to called add */
        U8      clg_off;                /* offset to callng add */
        U8      tcap_off;               /* offset to TCAP data  */
        U8      op_off;                /* offset to optional data field */

                                            /*  for above fields    */
        U8      ud[CCITT_SC_USER_MAX - 2]; /* SCCP Header plus  */
                                            /* TCAP Header and data */
} sccp_xuser_t;
```

* `msg_type` (output)
  Specifies the message type for the SCCP message.

* `ret_prot` (output)
  Specifies the protocol class to use when sending the MSU and the return option which specifies the action to take if an error occurs. You use a single value to define both parameters, as shown in the following chart:

Value	Description
0	Connectionless Class 0, no return on error.
1	Connectionless Class 1, no return on error.
0 x 80	Connectionless Class 0, returns an error.
0 x 81	Connectionless Class 1, returns an error.

* `hop_counter(`output`)`
  The value of the hop counter is decremented on each global title translation and should be in range 15 to 1.

* `cld_off` (output)
  Specifies the offset to the called address for the SCCP message.

* `clg_off` (output)
  Specifies the offset to the calling address for the SCCP message.

* `tcap_off` (output)
  Specifies the offset to the data portion of the SCCP message.

* `ud[SC_USER_MAX]` (output)
  Specifies the SCCP message.

The `iblk_t` Structure

The `iblk_t` structure contains the following fields and is defined in the include file `mblock.h`.

```
typedef struct iblk_s
{
        i_block_t iblk;
        U8        data[MAX_MBLK_DATA - sizeof(i_block_t)];
} iblk_t;
```

* `iblk` (input)
  This field (which specifies the `I_Block` structure) is internal to the SINAP/SS7 system and you should not modify it. See the include file `iblock.h` for a description of the `i_block_t` structure.

* `data` (input)
  This field is internal to the SINAP/SS7 system and you should not modify it.

FILES

`arch.h`, `ca_error.h`, `iblock.h`, `mblock.h`, `sinap.h`, `sysdefs.h`, `sys/time.h`, `timestamp.h`

RETURN VALUES

The `ca_put_msu()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful.

Possible UNIX values for `errno` are as follows.

Value	Meaning
`EBADF`	An invalid open file descriptor was specified.
`ENOTTY`	This `fides` is not associated with a device driver that accepts control functions.
`EIO`	An I/O error occurred during a read or write operation.
`EFAULT`	The pointer to the specified message is outside the address space allocated to the process.
`EINTR`	A signal was caught during the read or system call.
`ENOMEM`	Kernel queue is full; try again.

Possible CASL values for `errno` are as follows.

Value	Meaning
`CA_ERR_ACCESS`	The process calling `ca_put_msu()` is not registered. Call `ca_register()` before calling this function.
`CA_ERR_MSU_CALLS`	The process calling `ca_put_msu()` is not registered for this service, or the service is not allowed.
`CA_ERR_NO_SS7_SVC`	The process calling `ca_put_msu()` is not registered for SS7 service. Reregister the process using `ss7_primitive=SS7_CTRL_PRIMITIVE` and `fss7=1`.

SEE ALSO

`ca_flush_msu()`, `ca_get_msu()`

Connection-Oriented Functions

The CCITT and the China network variants support connection-oriented services. The SINAP/SS7 system uses the following CASL functions for implementing connection-oriented services.

- The `ca_get_sc()` function retrieves a connection-oriented message from a remote application.
- The `ca_put_sc()` function sends a connection-oriented message to a remote application.

This section also contains detailed information on the structures an application must initialize when sending either an IPC message to the SCCP-SCOC process or a data MSU to another application. The following structures are described:

- `sccp_ipc_t` - Passes IPC messages between the local application and the SCCP-SCOC process. This structure contains several structures, each of which passes a particular type of message.
- `sccp_prim_t` - An internal structure that conveys information about large messages, such as the message size and buffer location.
- `sccp_cldclg_t` - Contains information about the SCCP called- or calling-party address for a connection-oriented message.
- `sccp_dt1_t` - Transports a data-form-1 message.
- `sccp_dt2_t` - Transports a data-form-2 message.
- `sccp_expdata_t` - Transports a message containing expedited data.

When sending either an IPC message to the SCCP-SCOC process or a data MSU to another application, your application must initialize the appropriate structures. When your application is processing an incoming MSU or IPC message, the structures will have been initialized by the other application, the SCCP-SCOC process, or the SINAP/SS7 system.

ca_get_sc()

SYNOPSIS

```
m_block_t     *ca_get_sc(
                  BOOL    fwait);
```

DESCRIPTION

The `ca_get_sc()` function, used in connection-oriented services, retrieves an incoming message from the remote application with which a connection has been established. The function retrieves the next available message from the application's inbound queue and returns a pointer to the message.

The `ca_get_sc()` function automatically reassembles large messages (that is, messages that are 257 to 8192 bytes in length). The function retrieves the user data from **all** of the MSUs in the message, stores the data in one of the memory buffers allocated to handle large messages, and returns a pointer to that memory buffer.

See "Connection-Oriented Services" in Chapter 3 for detailed descriptions of the structures used in connection-oriented messages.

PARAMETERS

* `fwait` (input)

Specifies whether the function call should wait for an MSU before returning.
Specify `1` if you want the function to wait for an MSU. The function will not return until it reads an MSU from the queue. Specify `0` if you want the function to return if there are no MSUs. The function returns the error `ENODATA` when there are no MSUs.

RETURN VALUES

If successful, the function call returns either of the following, depending on the message's size.

- For messages of 256 bytes or less, the function returns a pointer to an `m_block_t` structure that contains the message.
- For messages that are 257 to 8192 bytes in length, the function returns a pointer to the memory buffer containing the message.

If an error occurs, the function call returns the value `-1` and `errno` is set to one of the following messages.

Error Message	Description
`CA_ERR_REG_NSCCP23`	The application did not register for connection-oriented services; therefore, it cannot call `ca_get_sc()`.
`CA_ERR_GETSC_BUSY`	No more connection IDs are available.
`CA_ERR_GETSC_SIZE`	The incoming MSU's user data is larger than the size of the memory buffer used for storing the data. Note that the function call returns the portion of the MSU user data that fits in the memory buffer and discards the rest of the user data.

ca_put_sc()

SYNOPSIS

```
int     ca_put_sc(
          m_block_t    *p_m);
```

DESCRIPTION

The `ca_put_sc()` function, used in connection-oriented services, sends an outgoing message to the remote application with which a connection has been established.

The `ca_put_sc()` function automatically performs message segmentation for large messages (that is, messages that are 257 to 8192 bytes in length). To send a large message, the application must write the message to a memory buffer and then pass `ca_put_sc()` a pointer to this buffer. The `ca_put_sc()` function then performs the necessary message segmentation, writing the message to several MSUs for transmission over the SS7 network.

The application must allocate memory for the buffer and then deallocate the memory.

See "Connection-Oriented Services" in Chapter 3 for detailed descriptions of the structures used in connection-oriented messages.

PARAMETERS

* `*p_m` (input)
Specifies a pointer to the message to be sent to the remote application.

- For messages of 256 bytes or less, this parameter is a pointer to the `m_block_t` structure that contains the message.
- For messages that are 257 to 8192 bytes in length, this parameter is a pointer to the memory buffer that contains the message. The application must allocate memory and initialize the buffer.

RETURN VALUES

If successful, the function call returns `0`.

If an error occurs, the function call returns the value `-1` and `errno` is set to one of the following messages.

Error Message	Description
`CA_ERR_REG_NSCCP23`	The application did not register for connection-oriented services; therefore, it cannot call `ca_put_sc()`.
`CA_ERR_PUTSC_NG`	The specified message type is invalid (for example, you attempted to send expedited data over a class-2 connection).
`CA_ERR_PUTSC_BAD`	The data in the `m_block_t.sccp_ctrl` structure is invalid.
`CA_ERR_PUTSC_SIZE`	The size of the MSU user data is invalid.
`CA_ERR_PUTSC_BUSY`	No more connection IDs are available.
`CA_ERR_PUTSC_CID`	The specified connection ID in the MSU is invalid.
`CA_ERR_PUTSC_MSG`	The application process is not registered as a control process; therefore, it cannot handle large messages.
`CA_ERR_PUTSC_CONN`	The connection ID has been lost; therefore, the SINAP/SS7 system cannot send the MSU.

Connection-Oriented Structures

This section contains detailed information on the structures an application must initialize when sending either an IPC message to the SCCP-SCOC process or a data MSU to another application.

In the descriptions of structure fields in the following sections, *input* indicates a field (and possibly a corresponding structure) that your application must initialize, and *output* indicates a field (and possibly a corresponding structure) that will have been initialized by the other application, SCCP-SCOC, or the SINAP/SS7 system.

The `sccp_ipc_t` Structure

The `sccp_ipc_t` structure passes IPC messages between a local application and the SCCP-SCOC process. The `sccp_ipc_t` structure is composed of several substructures, each of which passes a particular type of IPC message.

The local application and the SCCP-SCOC process each initiate different types of IPC messages. For example, it is the local application that issues a connection-request message

(`I_N_CONNECT_REQ`); therefore, the application is responsible for initializing the `sccp_ipc_t.scoc_con_req_t` structure and setting the `sccp_ipc_t` structure's `n_connect_req` field to point to the initialized `scoc_con_req_t` structure.

Likewise, it is the SCCP-SCOC process that issues an `I_SCOC_CID_RESULT` message in response to the local application's request for a connection ID. Therefore, SCCP-SCOC must initialize the `sccp_ipc_t.scoc_cid_result_t` structure to define the connection ID. SCCP-SCOC must also initialize the `sccp_ipc_t` structure's `cid_result` field to point to the initialized `scoc_cid_result_t` structure.

The following sample shows the format of the `sccp_ipc_t` structure. The `sccp_ipc_t` structure and all of the structures within it are defined in the `scoc-prims.h` include file.

```
typedef struct sccp_ipc_s
{
iblock_t     iblock;

union {
         scoc_con_req_t        n_connect_req;
         scoc_con_ind_t        n_connect_ind;
         scoc_con_res_t        n_connect_res;
         scoc_con_con_t        n_connect_con;
         scoc_res_req_t        n_reset_req;
         scoc_res_ind_t        n_reset_ind;
         scoc_res_res_t        n_reset_res;
         scoc_res_con_t        n_reset_con;
         scoc_dis_req_t        n_disconnect_req;
         scoc_dis_ind_t        n_disconnect_ind;
         scoc_inf_req_t        n_inform_req;
         scoc_inf_ind_t        n_inform_ind;
         scoc_get_connid_t     get_connid;
         scoc_cid_result_t     cid_result;
         scoc_rel_lrn_t        n_rel_lrn_req;
         scoc_timer_t          timer;
         scoc_change_t         change;
         scoc_change_ack_t     chgack;
      } primitives;
} sccp_ipc_t;
```

The following list describes the fields in the `sccp_ipc_t` structure. For more information about the IPC message to which each structure corresponds, see "Connection-Oriented Control Primitives Used in IPC Messages" and "Connection-Oriented Data Primitives Used in Data MSUs" in Chapter 2.

* `iblock` (input-output)
  The name of an `i_block_t` structure that contains information for the IPC message.

* `n_connect_req` (input)
  The name of an `scoc_con_req_t` structure that contains the connection request.

* `n_connect_ind` (output)
  The name of an `scoc_con_ind_t` structure that contains the connection-request indication.

* `n_connect_res` (input)
  The name of an `scoc_con_res_t` structure that contains the connection-request response.

* `n_connect_con` (output)
  The name of an `scoc_con_con_t` structure that contains the connection-request confirmation.

* `n_reset_req` (input)
  The name of an `scoc_res_req_t` structure that contains the connection-reset request.

* `n_reset_ind` (output)
  The name of an `scoc_res_ind_t` structure that contains the connection-reset indication.

* `n_reset_res` (input)
  The name of an `scoc_res_res_t` structure that contains the connection-reset response.

* `n_reset_con` (output)
  The name of an `scoc_res_con_t` structure that contains the connection-reset confirmation.

* `n_disconnect_req` (input)
  The name of an `scoc_dis_req_t` structure that contains the disconnect request.

* `n_disconnect_ind` (output)
  The name of an `scoc_dis_ind_t` structure that contains the disconnect-request indication.

* `n_inform_req` (input)
  The name of an `scoc_inf_req_t` structure that contains the information request. This message type currently is not supported; therefore, you should not modify this field.

* `n_inform_ind` (output)
  The name of an `scoc_inf_ind_t` structure that contains the information-request indication. This message type currently is not supported; therefore, you should not modify this field.

* `get_connid` (input)
  The name of an `scoc_get_connid_t` structure that contains the connection ID request.

* `cid_result` (output)
  The name of an `scoc_cid_result_t` structure that contains a connection ID. SCCP-SCOC provides this information in response to the local application's request for a connection ID.

* `n_rel_lrn_req`
  The name of an `scoc_rel_lrn_t` structure that contains a request to release LRNs. This message type currently is not supported; therefore, you should not modify this field.

* `timer`
  The name of an `scoc_timer_t` structure that contains a request to access a connection-oriented-services timer. This message type currently is not supported; therefore, you should not modify this field.

* `change`
  The name of an `scoc_change_t` structure that contains a change request. This message type currently is not supported; therefore, you should not modify this field.

* `chgack`
  The name of an `scoc_change_ack_t` structure that contains an acknowledgment for a change request. This message type currently is not supported; therefore, you should not modify this field.

The `sccp_prim_t` Structure

The `sccp_prim_t` structure conveys information about large messages. The structure is defined in the `mblock.h` include file. The following sample shows the structure's format.

```
typedef struct    sccp_prim_s
{
        U16  conn_id;                   /* sccp connection id */
        U8   more_data_ind;             /* sccp class 2 & 3 msgs, 0= no */
                                /* more data, 1 = more data */
        U8   flow_control;              /* flow control inform indication */
        U16  user_data_size;            /* size of user data */
        U8   copy_to_scoc;              /* set to indicate this msu should */
                                        /* copied to SCOC as well as sent */
                                        /* to the user */

        U8   grey_book;                 /* NEC grey book connection */
        U8   *p_user_data;              /* ptr to user data buffer  */
#ifdef _LP_32_64_
        U32  filler;                  /* For User32/Driver64 compatibility */
#endif  /* _LP_32_64_ */

} sccp_prim_t;
```

The following list describes the fields in the `sccp_prim_t` structure.

> **NOTE**
>
> The `sccp_prim_t` structure's `*p_user_data` and `user_data_size` fields are used for processing large messages. All other fields are internal to the SINAP/SS7 system and should not be modified.

* `conn_id` (input-output)
  The connection ID for the message.

* `more_data_ind` (output)
  Indicates whether the message contains more data than can fit in a single MSU. If the message has additional data, the value of this field is `1`; otherwise, it is `0`.

* `flow_control` (input-output)
  Indicates whether the application is implementing flow control. If flow control is being implemented, the value of this field is `1`; otherwise, it is `0`.

* `*p_user_data` (input-output)
  A pointer to a memory buffer that contains the message data. To send a large message, your application must set this field to point to a memory buffer in which the data is stored. The application must also allocate memory for the buffer and initialize it.

* `user_data_size` (input-output)
  The size of the message data. To send a large message, your application must set this field to indicate the data's length.

* `copy_to_scoc` (output)
  Indicates whether the message (`SC_EXPEDITED_DATA`, `SC_RESET_REQUEST`, or `SC_RELEASED`) must be copied to SCCP-SCOC so that SCCP-SCOC can respond to the remote application with an acknowledgment. If the message must be copied to SCCP-SCOC, the value of this field is `1`; otherwise, it is `0`.

* `grey_book` (output)
  Indicates whether the application is implementing the NEC grey-book feature. If the application is implementing the NEC grey-book feature, the value of this field is `1`; otherwise, it is `0`.

The `sccp_cldclg_t` Structure

The `sccp_cldclg_t` structure defines SCCP called- or calling-party address information for an MSU passed between local and remote applications. The structure is defined in the `mblock.h` include file. The following sample shows the structure's format.

```
typedef struct sccp_cldclg_s
{
        U32     pc;
        U8      pc_ind;
        U8      ssn;
        U8      ssn_ind;
        U8      gti_len;
        U8      gti_ind;
        U8      rtg_ind;
        U8      national;
        U8      gti[MAX_GTI_LEN];
} sccp_cldclg_t;
```

The following list describes the fields in the `sccp_cldclg_t` structure.

* `pc` (input-output)
  The point code of the remote (called) application or the local (calling) application.

* `pc_ind` (input-output)
  Indicates whether the SCCP called- or calling-party address contains a point code. The value `1` indicates the presence of a point code; otherwise, the value of this field is `0`.

* `ssn` (input-output)
  The SSN of the remote (called) application or the local (calling) application.

* `ssn_ind` (input-output)
  Indicates whether an SSN is included in the SCCP called- or calling-party address. The value `1` indicates the presence of an SSN; otherwise, the value of this field is `0`.

* `gti_len` (input-output)
  The length of the global title defined in the `gti` field. The value `0` indicates that the SCCP called- or calling-party address does not contain a global title. (See Chapter 3 for more information about global titles.)

* `gti_ind` (input-output)
  The format of the global title included in the SCCP called- or calling-party address. This field defines bits 6 through 3 of the address indicator. This field is blank if the SCCP called- or calling-party address does not contain a global title. (See Chapter 3 for more information about global titles.)

* `rtg_ind` (input-output)
  The routing-indicator bit of the address-indicator portion of the SCCP called- or calling-party address. The routing-indicator bit specifies the type of routing used to route the MSU to its destination: the value `0` indicates routing on global title, and the value `1` indicates routing on DPC and SSN.

* `national` (input-output)
  This field is used for national networks. It is always set to `0`.

* `gti[MAX_GTI_LEN]` (input-output)
  The global title included in the SCCP called- or calling-party address. This field is blank if the SCCP called- or calling-party address does not contain a global title. (See Chapter 3 for more information about global titles.)

The `sccp_dt1_t` Structure

The `sccp_dt1_t` structure defines a data-form-1 message. The structure is defined in the `sccphdrs.h` include file. The following sample shows the structure's format.

```
typedef struct sccp_dt1_s
{
     U8    msg_type;
     U8    dest_lrn[3];
     U8    seg_reass;
     U8    var_ptr;
     U8    ud_len;
     U8    ud[256];       /* data is 2 to 256 bytes */
} sccp_dt1_t;
```

The following list describes the fields in the `sccp_dt1_t` structure.

* `msg_type` (input-output)
  The MSU's message type. The only valid value for this field is `SC_DATA_FORM1`.

* `dest_lrn[3]` (input-output)
  The destination LRN for the MSU. This field is internal to the SINAP/SS7 system; you should not modify it.

* `seg_reass` (input-output)
  Indicates whether the message requires segmentation and reassembly. This field is internal to the SINAP/SS7 system; you should not modify it.

* `var_ptr` (input-output)
  This field is internal to the SINAP/SS7 system; you should not modify it.

* `ud_len` (input-output)
  The length of the MSU user data defined in the `ud` field. The value `0` indicates that the MSU does not contain user data.

* `ud[256]` (input-output)
  MSU user data, which can be up to 256 bytes in length. This field is blank if the MSU contains no user data.

The `sccp_dt2_t` Structure

The `sccp_dt2_t` structure defines a data-form-2 message. The structure is defined in the `sccphdrs.h` include file. The following sample shows the structure's format.

```
typedef struct sccp_dt2_s
{
     U8    msg_type;
     U8    dest_lrn[3];
     U8    seq_seg[2];
     U8    var_ptr;
     U8    ud_len;
     U8    ud[256];       /* data is 2 to 256 bytes */
} sccp_dt2_t;
```

The following list describes the fields in the `sccp_dt2_t` structure.

* `msg_type` (input-output)
  The MSU's message type. The only valid value for this field is `SC_DATA_FORM2`.

* `dest_lrn[3]` (input-output)
  The destination LRN for the MSU. This field is internal to the SINAP/SS7 system; you should not modify it.

* `seq_seg[2]` (input-output)
  This field is internal to the SINAP/SS7 system; you should not modify it.

* `var_ptr` (input-output)
  This field is internal to the SINAP/SS7 system; you should not modify it.

* `ud_len` (input-output)
  The length of the MSU user data defined in the `ud` field. The value `0` indicates that the MSU does not contain user data.

* `ud[256]` (input-output)
  MSU user data, which can be up to 256 bytes in length. This field is blank if the MSU contains no user data.

The `sccp_expdata_t` Structure

The `sccp_expdata_t` structure defines a message containing expedited data. The structure is defined in the `sccphdrs.h` include file. The following sample shows the structure's format.

```
typedef struct sccp_expdata_s
{
     U8    msg_type;
     U8    dest_lrn[3];
     U8    var_ptr;
     U8    ud_len;
     U8    ud[32];       /* data is 2 to 32 bytes */
} sccp_expdata_t;
```

The following list describes the fields in the `sccp_expdata_t` structure.

* `msg_type` (input-output)
  The MSU's message type. The only valid value for this field is `SC_EXPEDITED_DATA`.

* `dest_lrn[3]` (input-output)
  The destination LRN for the MSU. This field is internal to the SINAP/SS7 system; you should not modify it.

* `var_ptr` (input-output)
  This field is internal to the SINAP/SS7 system; you should not modify it.

* `ud_len` (input-output)
  The length of the MSU user data defined in the `ud` field. The value `0` indicates that the MSU does not contain user data.

* `ud[32]` (input-output)
  MSU user data, which can be up to 32 bytes in length. This field is blank if the MSU contains no user data.

TCAP Functions

This section contains an alphabetic reference of the following CASL functions, which are used in applications that interface to the SS7 network at the TCAP boundary.

- `ca_alloc_tc()`
- `ca_cust_dist_cmd()` (CCITT/ANSI)
- `ca_dealloc_tc()`
- `ca_dec_cs1_corrid( )` (CCITT/ANSI)
- `ca_dist_cmd()`
- `ca_enc_cs1_corrid( )` (CCITT/ANSI)
- `ca_get_dial_id()` (CCITT/China)
- `ca_get_tc()`
- `ca_get_trans_id()` (ANSI)
- `ca_process_tc()`
- `ca_put_tc()`
- `ca_rel_dial_id()` (CCITT/China)
- `ca_rel_trans_id()` (ANSI)

ca_alloc_tc()

SYNOPSIS

```
int     ca_alloc_tc();
```

DESCRIPTION

The `ca_alloc_tc()` function allocates a `T_Block` for the client application process. The `ca_alloc_tc()` function returns an index into the `T_Block` array for the next available `T_Block` entry. If no entries are available, `ca_alloc_tc()` returns an error indication.

`T_Blocks` communicate TCAP components between the CASL functions and client application processes. When a client application process registers, an array of a specified number of `T_Blocks` is created in the client process's data space. The `T_Block` array is a working area for both the CASL and the client application.

As MSUs are received and decoded, individual TCAP components are placed in a `T_Block`, and the index into the `T_Block` array is returned in response to each `ca_get_tc()` function call that the client process requests. The client can modify a specified `T_Block` and pass it back, or make it available for use again (deallocate it). The `ca_alloc_tc()` function gets the next available entry from the `T_Block` free pool, inserts the value of `owner_id` in the `T_Block` owner ID table, and returns the index of `T_Block` to the user. If the application is not registered at the TCAP boundary, `ca_alloc_tc()` returns an error.

The client can also allocate one or more `T_Block` entries for its own use. If the client passes these entries to the other TCAP functions, it assumes the TCAP will deallocate them. If the client does not pass these entries for output, it must return them to the available pool by calling `ca_dealloc_tc()`.

FILES

`$SINAP_HOME/Include/ca_error.h`

RETURN VALUES

The `ca_alloc_tc()` function returns an index to the next available `T_Block`. If there is an error, the function returns a `-1`. CASL values for `errno` are defined in `ca_error.h`. UNIX values are defined in `sys/errno.h`.

The TCAP can return the following values.

Value	Action
TC_ERR_NOT_REG_AT_TCAP_BOUNDARY	Reregister the application at the TCAP boundary using SS7_input_boundary= SS7_INPUT_BOUNDARY_TCAP.
TC_ERR_T_BLOCK_CAPACITY_EXHAUSTED	Reregister the application with a greater number of T_BLOCKs.

SEE ALSO

ca_dealloc_tc(), ca_get_tc(), ca_get_trans_id(), ca_process_tc(), ca_put_tc(), ca_rel_trans_id()

ca_dealloc_tc()

SYNOPSIS

```
int     ca_dealloc_tc(
          S32 tb_index);
```

DESCRIPTION

The `ca_dealloc_tc()` function deallocates a `T_Block` and returns it to the free pool for reallocation. If a client application process is not registered at the TCAP boundary, `ca_dealloc_tc()` returns an error.

For incoming messages, the application calls `ca_dealloc_tc()` to deallocate the `T_Block` after calling the `ca_get_tc()` function. For outgoing messages, the CASL calls `ca_dealloc_tc()` after extracting information from the `T_Block`.

PARAMETERS

* `tb_index` (input)
  Specifies an index into the `T_Block` array for the `T_Block` being deallocated.

FILES

`$SINAP_HOME/arch.h`, `ca_error.h`

RETURN VALUES

The `ca_dealloc_tc()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

The TCAP can return the following errors.

Error	Action
TC_ERR_NOT_REG_AT_TCAP_BOUNDARY	Reregister the application at the TCAP boundary using ss7_input_boundary= SS7_INPUT_BOUNDARY_TCAP.
TC_ERR_INV_T_BLOCK_INDEX	The T_Block index is out of range. Make sure the application is using the correct T_Block index.
TC_ERR_T_BLOCK_NOT_ALLOCATED	The T_Block has already been deallocated.
TC_ERR_TCAP_OWN_T_BLOCK	The T_Block is under the TCAP's control and no further action required. If this T_Block must be released, the application can issue a TC_U_CANCEL with the appropriate transaction and invoke IDs.

SEE ALSO

ca_alloc_tc(), ca_get_trans_id(), ca_get_tc(), ca_process_tc(), ca_put_tc(), ca_rel_trans_id()

ca_dist_cmd()

SYNOPSIS

```
int ca_dist_cmd (dist_cmd_t *ssn_opc_table);
```

DESCRIPTION

The `ca_dist_cmd()` function defines message-distribution information for an application, or it modifies or deletes an application's existing message-distribution information. This function has one parameter, *`*ssn_opc_table`*, which is a pointer to a `dist_cmd_t` structure that defines such information as the application's message-distribution information (`ssn_count`, `opc_count`, `ssn`, and `opc`) and the type of action the function is to perform (`cmd`).

The `dist_cmd_t` structure's `ssn` and `opc` fields are arrays in which you define the discrimination rules that you want the SINAP/SS7 system to use to route incoming MSUs to the application. the SINAP/SS7 system compares an incoming MSU with the application's message-distribution information. If the characteristics of the MSU match the application's message-distribution information, the SINAP/SS7 system passes the MSU to the application; otherwise, the SINAP/SS7 system discards the MSU.

NOTE

If you are using the custom application distribution (CAD) feature which extends the capabilities of the enhanced message distribution feature, call the `ca_cust_dist_cmd()` function to define message distribution criteria for an application.

PARAMETERS

The `ca_dist_cmd()` function has a single parameter, a pointer to the structure `dist_cmd_t (d`efined in the include file `register.h)`, which you must initialize before calling `ca_dist_cmd()`. The structure has the following format. (See the r`egister.h` include file for the definitions of `MAX_APPL_SSN` and `MAX_APPL_OPC`.)

The `dist_cmd_t` Structure

The `dist_cmd_t` structure defines an application's message distribution information.

```
typedef struct dist_cmd_s
{
      U32 appl;                /* APPL_THIS -1 */
      U8  cmd;                 /* DIST_SET 1 */
                               /* DIST_DEL 2 */
                               /* DIST_INQ 3 */
      U8  boundary;            /* SS7_INPUT_BOUNDARY_NA 0 */
                          /* SS7_INPUT_BOUNDARY_SCCP23 4 */
      S8  ssn_count;           /* DIST_ALL 0 */
      U8  opc_count;           /* DIST_ALL_OTHER 0 */
      U8  ssn[MAX_APPL_SSN];   /* 32 */
      U32 opc[MAX_APPL_OPC];   /* 256 */
} dist_cmd_t;
```

The `dist_cmd_t` structure contains the following fields:

* `appl` (input)
  Specifies specifies the name of the application whose message-distribution information you want to define or retrieve.

  NOTE

  In the `appl` field of the `register_req_t` (`CA_REG`) structure, the application name is specified as an ASCII character string of up to four bytes. However, in the `appl` field of the `dist_cmd_t` structure, the application name must be specified as a zero-filled, **right-justified** U32 word. To convert an application name to this format, code the application so that it calls the function `ca_pack()`, passing the character string that defines the application name; the function returns the application name as a zero-filled, right-justified U32 word. To convert the application name back to a character string, code the application so that it calls the function `ca_unpack()`, passing the U32 word and a pointer to a character string; the function converts the application name to a character string.

* `cmd` (input)
  Specifies the task to perform on the application's message-distribution information. Valid values are as follows:

  - `DIST_SET` uses the information in the `ssn` and `opc` fields to define the message-distribution information for an application or to modify an application's existing message-distribution information.

- `DIST_DELETE` deletes an application's message-distribution information, which means that the application no longer supports enhanced message distribution.
- `DIST_INQ` retrieves an application's message-distribution information.

* `boundary` (input)
Specifies the application boundary, either `SS7_INPUT_BOUNDARY_NA` for a non-COF application or `SS7_INPUT_BOUNDARY_SCCP23` for a COF application. Only required for a `DIST_INQ command` with the `appl` field = 0.

* `ssn_count` (input)
Specifies the number of entries in the application's SSN array (`ssn`). The value you specify cannot exceed the value of `MAX_APPL_SSN`, which is defined in the include file `$SINAP_HOME/Include/register.h`.

If you do not want the SINAP/SS7 system to perform message discrimination based on an incoming MSU's SSN, specify the value `0` for `ssn_count` and leave the `ssn` field empty.

* `opc_count` (input)
Specifies the number of entries in the application's OPC array (`opc`). The value you specify cannot exceed the value of `MAX_APPL_OPC`, which is defined in the include file `$SINAP_HOME/Include/register.h`.

If you do not want the SINAP/SS7 system to perform message discrimination based on an incoming MSU's OPC, specify `0` for `opc_count` and leave the `opc` field empty. In this case, the SINAP/SS7 system sends the application all incoming MSUs destined for it regardless of the MSU's OPC, provided that no other application is registered for the MSU's OPC.

* `ssn[MAX_APPL_SSN]` (input)
Is an array of SSNs to be associated with the application. The number of entries in this array must match the value defined by `ssn_count`. If you specified `0` for `ssn_count`, make sure that the array is empty. (`MAX_APPL_SSN` is defined in the `register.h` include file.)

When the SINAP/SS7 system receives an incoming MSU, it examines the MSU's SSN; if the SSN is listed in this array, the SINAP/SS7 system sends the MSU to the application. (For example, if you associate the SSNs `220`, `230`, and `240` with the application, the SINAP/SS7 system sends the application all of the MSUs addressed to SSNs `220`, `230`, and `240`.)

* `opc[MAX_APPL_OPC]` (input)
An array of originating point codes (OPCs) from which the application can accept incoming MSUs. The number of entries in this array must match the value defined by *`opc_count`*. If you specified `0` for `opc_count`, make sure that this array is empty. (`MAX_APPL_OPC` is defined in the SINAP/SS7 `register.h` include file.)

When the SINAP/SS7 system receives an incoming MSU destined for the application, it examines the MSU's OPC; if the OPC is listed in this array, the SINAP/SS7 system sends the MSU to the application; otherwise, it discards the MSU. Note, however, that if you defined the environment variable `UDTS_NO_OPC`, the SINAP/SS7 system does not discard the MSU, but instead sends a UDTS message to the OPC.

INCLUDE FILES

`$SINAP_HOME/Include/ca_glob.h,ca_error.h,arch.h`

RETURN VALUES

The `ca_dist_cmd` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

The CASL can return the following errors:

Error	Action
`CA_ERR_ACCESS`	Register the application with CASL.
`CA_ERR_NULL_DIST`	Call the function with a non-NULL pointer.
`CA_ERR_NULL_APPL`	Use a nonzero application ID.
`CA_ERR_DIST_CMD`	Use a valid command code.or boundary
`CA_ERR_NEG_COUNT`	Use a non-negative `ssn_count` or `opc_count`.
`CA_ERR_MAX_COUNT`	Use a value lower than the maximum for `ssn_count` or `opc_count`.

NOTES

The man page format of this command is `ca_dist_cmd`.

ca_cust_dist_cmd()

SYNOPSIS

```
int ca_cust_dist_cmd (cust_dist_cmd_t *custom_table);
```

DESCRIPTION

The `ca_cust_dist_cmd()` function is an extended version of the `ca_dist_cmd()` function used in enhanced distribution. This function defines message-distribution information for an application, or it modifies or deletes an application's existing message-distribution information. In addition to specifying the application name, command, SSN, and OPC criteria, the function also specifies the ID of the type of custom distribution being implemented and the type-specific structure used to define the custom distribution criteria. The type-specific criteria structure is specified as an abstract (`void *`) pointer. The actual structure type is determined by the custom type ID parameter.

The `cust_dist_cmd_t` structure's `ssn` and `opc` fields are arrays in which you define the discrimination rules that you want the SINAP/SS7 system to use to route incoming MSUs to the application. The SINAP/SS7 system compares an incoming MSU with the application's message-distribution information. If the characteristics of the MSU match the application's message-distribution information, the SINAP/SS7 system passes the MSU to the application; otherwise, the SINAP/SS7 system discards the MSU.

PARAMETERS

The `ca_cust_dist_cmd()` function has a single parameter, an abstract (`void*`) pointer to the structure `cust_dist_cmd_t`, which you must initialize before calling `ca_cust_dist_cmd()`. This structure is defined in the include file `cust_dist.h` and has the following format. (See the `register.h` include file for the definitions of `MAX_APPL_SSN` and `MAX_APPL_OPC`.)

The `cust_dist_cmd_t` Structure

The `cust_dist_cmd_t` structure is used to specify the application name, SSN, OPC, and the ID of the type of custom distribution being implemented as well as a type specific structure to

specify the custom distribution criteria. The `cust_dist_cmd_t` is composed of three substructures that are described in the sections following the `cust_dist_cmd_t` structure.

```
typedef struct cust_dist_cmd_s
{
   dist_cmd_t        ssn_opc_table;
   cust_dist_id_t    custom_id;
   union
   {
       cs1_inap_v01_tbl_t    cs1_inap_v01;
} cust_dist_cmd_t;
```

The `dist_cmd_t` Structure

The `dist_cmd_t` structure defines an application's message distribution information.

```
typedef struct dist_cmd_s
{
      U32 appl;                /* APPL_THIS -1 */
      U8  cmd;                 /* DIST_SET 1 */
                               /* DIST_DEL 2 */
                               /* DIST_INQ 3 */
      U8  boundary;            /* SS7_INPUT_BOUNDARY_NA 0 */
                          /* SS7_INPUT_BOUNDARY_SCCP23 4 */
      S8  ssn_count;           /* DIST_ALL 0 */
      U8  opc_count;           /* DIST_ALL_OTHER 0 */
      U8  ssn[MAX_APPL_SSN];   /* 32 */
      U32 opc[MAX_APPL_OPC];   /* 128 */
} dist_cmd_t;
```

The `dist_cmd_t` structure contains the following fields:

* `appl` (input)
  Specifies specifies the name of the application containing the message-distribution information to be defined or retrieved.

  NOTE

  In the `appl` field of the `register_req_t` (`CA_REG`) structure, the application name is specified as an ASCII character string of up to four bytes. However, in the `appl` field of the `dist_cmd_t` structure, you must specify the application name as a zero-filled, **right-justified** U32 word. To convert an application name to this format, code the application so that it calls the function `ca_pack()`, passing the character string that defines the application name; the function returns the application name as a zero-filled, right-justified U32 word. To

convert the application name back to a character string, code the application so that it calls the function `ca_unpack()`, passing the U32 word and a pointer to a character string; the function converts the application name to a character string.

* `cmd` (input)
  Specifies the task to perform on the application's message-distribution information. Valid values are as follows:

  - `DIST_SET` uses the information in the ssn and opc fields to define the message-distribution information for an application or to modify an application's existing message-distribution information.
  - `DIST_DELETE` deletes an application's message-distribution information, which means that the application no longer supports enhanced message distribution.
  - `DIST_INQ` retrieves an application's message-distribution information.

* `boundary` (input)
  Specifies the application boundary, either `SS7_INPUT_BOUNDARY_NA` for a non-COF application or `SS7_INPUT_BOUNDARY_SCCP23` for a COF application. The latter is not supported for Custom Distribution. Only required for a `DIST_INQ command` with the `appl` field = 0.

* `ssn_count` (input)
  Specifies the number of entries in the application's SSN array (`ssn`). The value you specify cannot exceed the value of `MAX_APPL_SSN`, which is defined in the include file `$SINAP_HOME/Include/register.h`.

  If you do not want the SINAP/SS7 system to perform message discrimination based on an incoming MSU's SSN, specify the value `0` for `ssn_count` and leave the `ssn` field empty.

* `opc_count` (input)
  Specifies the number of entries in the application's OPC array (`opc`). The value you specify cannot exceed the value of `MAX_APPL_OPC`, which is defined in the include file `$SINAP_HOME/Include/register.h`.

  If you do not want the SINAP/SS7 system to perform message discrimination based on an incoming MSU's OPC, specify `0` for `opc_count` and leave the `opc` field empty. In this case, the SINAP/SS7 system sends the application all incoming MSUs destined for it regardless of the MSU's OPC, provided that no other application is registered for the MSU's OPC.

* `ssn[MAX_APPL_SSN]` (input)
  Is an array of SSNs to associate with the application. The number of entries in this array must match the value defined by `ssn_count`. If you specified `0` for `ssn_count`, make sure that the array is empty. `MAX_APPL_SSN` is defined in the `register.h` include file.

When the SINAP/SS7SINAP/SS7 system receives an incoming MSU, it examines the MSU's SSN. If the SSN is listed in this array, the SINAP/SS7 system sends the MSU to the application. For example, if you associate the SSNs 220, 230, and 240 with the application, the SINAP/SS7 system sends the application all of the MSUs addressed to SSNs 220, 230, and 240.

* `opc[MAX_APPL_OPC]` (input)
  An array of originating point codes (OPCs) from which the application can accept incoming MSUs. The number of entries in this array must match the value defined by *`opc_count`*. If you specified `0` for `opc_count`, make sure that this array is empty. `MAX_APPL_OPC` is defined in the SINAP/SS7 `register.h` include file.

  When the SINAP/SS7 system receives an incoming MSU destined for the application, it examines the MSU's OPC. If the OPC is listed in this array, the SINAP/SS7 system sends the MSU to the application, otherwise, it discards the MSU. Note, however, that if you defined the environment variable `UDTS_NO_OPC`, the SINAP/SS7 system does not discard the MSU, but instead sends a UDTS message to the OPC.

The `cust_dist_id_t` Structure

The `cust_dist_id_t` structure specifies the ID of the type of custom distribution being implemented.

```
typedef enum
{
     CS1_INAP_V01 = 1/* Only value currently supported */
} cust_dist_id_t;
```

The `cs1_inap_v01_tbl_t` Structure

The `cs1_inap_v01_tbl_t` structure specifies all ServiceKey parameters.

```
typedef struct cs1_inap_v01_tbl_s
{
   int          svc_key_count;
   int          svc_key [MAX_APPL_SVC_KEY];/*[64]*/
} cs1_inap_v01_tbl_t;
```

* `svc_key_count` (input)
  Specifies the number of ServiceKey services to be used. The value you specify cannot exceed the value of `MAX_APPL_SVC_KEY` defined in the include file `$SINAP_HOME/Include/cust_dist.h`.

  Note that a value of `0` specifies the application as the fallback (or default) application. All message traffic that passes the SSN and OPC criteria specified for this application, but cannot be sent to any other application registered for the same SSN/OPC with specific ServiceKey criteria, will be sent to this fallback application.

* `svc_key` (input)
  An array of up to 64 individual ServiceKeys. The number of entries must match the number defined in `svc_key_count`. If you specified 0 for `svc_key_count`, make sure that this array is empty. `MAX_APPL_SVC_KEY` is defined in the SINAP/SS7 `cust_dist.h` include file.

INCLUDE FILES

`$SINAP_HOME/Include/ca_glob.h,ca_error.h,arch.h,register.h, cust_dist.h`

RETURN VALUES

The `ca_cust_dist_cmd` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning. See `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

The CASL can return the following errors:

Error	Action
`CA_ERR_ACCESS`	Register the application with CASL.
`CA_ERR_NULL_DIST`	Call the function with a non-NULL pointer.
`CA_ERR_NULL_APPL`	Use a nonzero application ID.
`CA_ERR_DIST_CMD`	Use a valid command code.or boundary
`CA_ERR_NEG_COUNT`	Use a non-negative `ssn_count` or `opc_count`.
`CA_ERR_MAX_COUNT`	Use a value lower than the maximum for `ssn_count` or `opc_count`.
`CA_ERR_CUST_APPL_INQ`	Use the function `ca_appl_name_lkup()` for custom application distribution.
`CA_ERR_NULL_APPL_LKUP`	Use a non-NULL `appl_name_table`.
`CA_ERR_CUST_DIST_ID`	Use a valid `custom_id`.
`CA_ERR_CS1INAP_SVCKEY_CNT`	Use a valid `svc_key_count`.
`CA_ERR_CS1INAP_MAX_ENTRYS`	Use a value lower than the maximum for `SSN_OPC` or `svc_key`.

Error	Action
CA_ERR_CS1INAP_SET_FAILED	The maximum number of ServiceKeys specified for the specified SSN/OPC criteria was exceeded. 1. Use a lower value, or 2. Reduce the number of SSN/OPC criteria.
CA_ERR_CS1INAP_NO_ENTRY	Use a valid DIST_INQ value. (No entry matching criteria specified was found.)
CA_ERR_CS1INAP_INCONSISTENT	Restart the SINAP/SS7 system. If the error persists, reboot the system. (DIST_INQ - Table inconsistency found.)

NOTES

The man page format of this command is `ca_cust_dist_cmd`.

SEE ALSO

`ca_enc_cs1_corrid()`

`ca_dec_cs1_corrid()`

ca_enc_cs1_corrid()

SYNOPSIS

```
int ca_enc_cs1_corrid(char *param, int tid);
```

DESCRIPTION

The `ca_enc_cs1_corrid()` function encodes the ETSI/ITU-T CS-1 INAP `CorrelationID` digits in a format compatible with CS-1 INAP Custom Application Distribution `AssistRequestInstructions` (ARI) processing. This is intended for use in encoding `CorrelationID` parameters in the `EstablishTemporaryConnection` (ETC) operation.

The SINAP/SS7 implementation encodes the digits to include the Interprocess Communications (IPC) index of the process and the Transaction or Dialogue ID of the transaction sending the ETC. This allows the SINAP node to route the ARI to the appropriate process, and the process to correlate the ARI to the appropriate original transaction. The IPC index is encoded as fixed length, five-digit, zero filled string, and the Transaction ID is encoded as fixed length, 10-digit, zero filled string for a combined total of 15 digits. In the ETC Operation, the `CorrelationID` parameter is encoded in the ITU-T Q.763 Generic Digits parameter format.

Table 6-1. Map of Encoding CorrelationID to Generic Digits Parameter Format

Octet	8	7	6	5	4	3	2	1
1	Encoding			Type of Digits				
2	IPC Index BCD digit 2				IPC Index BCD digit 1			
3	IPC Index BCD digit 4				IPC Index BCD digit 3			
4	Transaction ID BCD digit 1				IPC Index BCD digit 5			
5	Transaction ID BCD digit 3				Transaction ID BCD digit 2			
6	Transaction ID BCD digit 5				Transaction ID BCD digit 4			
7	Transaction ID BCD digit 7				IPC Index BCD digit 6			
8	Transaction ID BCD digit 9				Transaction ID BCD digit 8			
9	Spare filler				Transaction ID BCD digit 10			

NOTE

This function does not perform the BCD encoding of the digit string. The application must perform final encoding of the `CorrelationID` Generic Digits parameter.

PARAMETERS

* `*param` (input)
  Specifies a pointer to a buffer that contains the `CorrelationID` digit string to be encoded. The digit string contains the IPC index of the process and the transaction/dialogue ID of the transaction sending the ETC. Addresses a buffer sized for [`10`] ASCII characters plus a `NULL` terminator (11).

* `tid` (input)
  Specifies the transaction/dialogue ID of the originator of the ETC, and the SINAP/SS7 IPC index of the process.

RETURN VALUES

This function returns the encoded CorrelationID digit string or `-1` on failure.

Value	Meaning
`Encoded Digits`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

SEE ALSO

`ca_dec_cs1_corrid()`

ca_dec_cs1_corrid()

SYNOPSIS

```
int ca_dec_cs1_corrid(char *param, int *tid, int *ipc_index);
```

DESCRIPTION

The `ca_dec_cs1_corrid()` function decodes the ETSI/ITU-T CS-1 INAP `CorrelationID` digits encoded in a format compatible with CS-1 INAP Custom Application Distribution `AssistRequestInstructions` (ARI) processing. This is intended for use in decoding `CorrelationID` parameters received in the ARI operation. The digits portion of the parameter are echoed from those originally encoded in the `EstablishTemporaryConnection` (ETC) operation. This allows the ARI to be correlated to the ETC.

The SINAP/SS7 implementation encodes the digits to include the IPC index of the process and the transaction/dialogue ID of the transaction sending the ETC. This allows the SINAP node to route the ARI to the appropriate process, and the process to correlate the ARI to the appropriate original transaction. The IPC index is encoded as fixed length, five-digit, zero filled string, and the Transaction ID is encoded as fixed length, 10-digit, zero filled string for a combined total of 15 digits. In the ARI operation, the `CorrelationID` parameter is encoded in the ITU-T Q.763 Generic Digits parameter format.

Table 6-2. Map of Decoding CorrelationID to Generic Digits Parameter Format

Octet	8	7	6	5	4	3	2	1
1	Number Qualifier Indicator							
2	odd	Nature of Address Indicator						
3		Numbering Plan			Pres Ind		Scrn Ind	
4	IPC Index BCD digit 2				IPC Index BCD digit 1			
5	IPC Index BCD digit 4				IPC Index BCD digit 3			
6	Transaction ID BCD digit 1				IPC Index BCD digit 5			
7	Transaction ID BCD digit 3				Transaction ID BCD digit 2			
8	Transaction ID BCD digit 5				Transaction ID BCD digit 4			
9	Transaction ID BCD digit 7				IPC Index BCD digit 6			
10	Transaction ID BCD digit 9				Transaction ID BCD digit 8			
11	Spare filler				Transaction ID BCD digit 10			

NOTE

This function does not perform the BCD decoding of the digit string. The `CorrelationID` Generic Digits parameter must first be decoded by the application. It is assumed that the digits have been converted to ASCII format.

PARAMETERS

* `*param` (input)
  Specifies a pointer to a buffer that contains the ASCII string of digits extracted from ITU-T Q.763 Generic Number parameter.

* `*tid` (output)
  Specifies a pointer to a buffer that contains the transaction/dialogue ID of the originator of the ETC.

* `*ipc_index` (output)
  Specifies a pointer to a buffer that contains the IPC index of the originator.

RETURN VALUES

This function returns the decoded `CorrelationID` digit string or `-1` on failure.

Value	Meaning
`Decoded Digits`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

SEE ALSO

`ca_enc_cs1_corrid()`

ca_get_dial_id()

SYNOPSIS

```
int      ca_get_dial_id();
```

DESCRIPTION

The `ca_get_dial_id()` function gets a unique dialogue ID for an application process before the process begins a dialogue. This function is used by the CCITT, TTC, NTT, and China network variants of the SINAP/SS7 system. The ANSI variant uses the function `ca_get_trans_id()`. For the SINAP/SS7 system to properly route the TCAP components that belong to a specific dialogue, the TCAP and TCAP user must adhere to the following dialogue assignments.

Per ITU-T (CCITT) Recommendations, the TCAP user assigns a dialogue ID when it initiates a new dialogue. The SINAP/SS7 TCAP assigns internal dialogue IDs (visible only to SINAP/SS7 processes and the SINAP/SS7 TCAP user) when a remote node initiates a dialogue via a `TR-UNI` or `TR-BEGIN` dialogue primitive. Because the TCAP uses dialogue IDs to access, store, and retrieve TCAP components, there is a potential problem when two separate functions assign dialogue IDs from the same pool of values. To accommodate dialogue ID assignment from the TCAP and TCAP user, there are two assigning paths: one for input and one for output.

On input, the TCAP user must issue a `ca_get_tc()` function call to receive the next available primitive (either dialogue or component). To support the need for dialogue IDs for itself and the TCAP user, the TCAP extends the dialogue ID in the `T_Block` to 32 bits and adds a parameter to the `ca_get_tc()` function call. Thus, the extended dialogue ID contains 16 bits for the TCAP and TCAP dialogue ID, and 16 bits for the TCAP user and TCAP user dialogue ID.

An additional `ca_get_tc()` parameter (`pfunc`) contains the address of the function the TCAP user wants TCAP to call when the remote node starts a new dialogue; the address must exist in each `ca_get_tc()` function call. TCAP calls this function only when a new dialogue is started. The TCAP call to the TCAP user's function provides an index parameter to the `T_Block` containing the `TC_BEGIN` or `TC_UNI` dialogue primitive and allows the TCAP user to examine the new dialogue primitive. The TCAP user updates its portion of the dialogue ID in the `T_Block` and returns control to the TCAP. The TCAP then assigns its own dialogue ID and returns it to the TCAP user. (This return is from the original `ca_get_tc()` call.)

The T_Block that ca_get_tc() returns has both the TCAP user and the TCAP dialogue ID, which the TCAP retains for future calls to ca_get_tc().

When a new dialogue begins on output, the TCAP user obtains a TCAP dialogue ID by calling ca_get_dial_id(). This TCAP user owns the dialogue ID and is the only user that can release it via a call to ca_rel_dial_id().

Once the TCAP user gets a dialogue ID, it obtains a T_Block by calling the ca_alloc_tc() function, and creates the TC_BEGIN or TC_UNI primitive in the T_Block. The TCAP user inserts the TCAP user and TCAP dialogue IDs in the T_Block dialogue ID field, so when the user issues the ca_put_tc() function call, both the TCAP and TCAP user recognize both parts of the dialogue ID.

All subsequent calls to ca_put_tc() must reference T_Blocks containing the TCAP dialogue ID. For debugging, the TCAP user should also place its dialogue ID in the field allocated for that purpose. The TC_REPLY and TC_CONTINUE primitives to a local TCAP user's dialogues from a remote node contain the TCAP user's dialogue ID.

The function does not have any parameters.

FILES

$SINAP_HOME/Include/ca_error.h

DIAGNOSTICS AND WARNINGS

The ca_get_dial_id() function returns a dialogue ID. If the function returns -1, there is an error. CASL values for errno are defined in ca_error.h; UNIX values are defined in sys/errno.h.

Value	Meaning
0	Successful.
-1	Unsuccessful. See errno for error number and meaning.

The TCAP can return the following errors.

Error	Action
TC_ERR_NOT_REG_AT_TCAP_BOUNDARY	Reregister the application at the TCAP boundary using ss7_input_boundary= SS7_INPUT_BOUNDARY_TCAP.

Error	Action
TC_ERR_OUT_OF_DIAL_ID	There are problems with the dialogue IDs. Either close dialogues with END, release dialogue IDs, or allocate more dialogue IDs and reregister.

SEE ALSO

ca_alloc_tc(), ca_dealloc_tc(), ca_get_tc(), ca_process_tc(), ca_put_tc(), ca_rel_dial_id()

ca_get_tc()

SYNOPSIS

```
int    ca_get_tc(
         BOOL    fwait,
         int     (*pfunc)(S32));
```

DESCRIPTION

The `ca_get_tc()` function returns an index to an inbound `T_Block`, which contains a TCAP component. This function is called whenever the application needs to receive a `TC_BEGIN` (CCITT), `TC_QRY_W_PERM` (ANSI), `TC_QRY_WO_PERM` (ANSI), or `TC_UNI` primitive. For information about the `T_Block`, see "The `T_Block` Structure (`t_block_t`)" later in this description.

To use this function, a client application must be registered to receive input at the TCAP boundary.

To control how TCAP assigns internal transaction IDs, you can create a user-defined function that assigns a unique transaction ID to a TCAP transaction. To do this, specify a pointer to a user-defined function in the `pfunc` parameter and pass it the `T_Block` index as an input parameter. Note that the user-defined function must assign a unique transaction ID to the `tc_user_id` field of the `T_Block`.

The TCAP starts a transaction timer before it allocates resources (such as the `T_Block`).

PARAMETERS

* `fwait` (input)
  Specifies whether the function waits for a TCAP component. Specify a value of `1` if you want the function call to execute in blocking mode (wait for input); otherwise, specify `0`. If you specify `0`, the function returns the error `CA_ERR_NO_MSUS` when there is no input.

* `pfunc` (input)
  Specifies a pointer to a user-supplied function. The `pfunc` parameter is typically used to set a `tc_user_id` in the `T_Block`. Specify `0` (or greater) for the return value if you want TCAP to retrieve an index to the next available `T_Block`. Values less than 1 for the return value from `pfunc` cause the error `TC_ERR_OUT_OF_TC_USER_ID` to be returned by `ca_get_tc` after `T_Block` deallocation.

If you want TCAP to call a user-defined function to assign transaction IDs to TCAP transactions, specify a pointer to that user-defined function as the value of this parameter. The input parameter to the user-defined function is the index of the `T_Block`.

The `T_Block` Structure (`t_block_t`)

For the `ca_get_tc()` function to work, you must set the following fields in the `t_block_t` structure, which is defined in the include file `tblock.h`.

```
 typedef struct t_block_s
{
          U8       primitive_type;
          U8      primitive_code;
#if defined(__LP64__) || defined(_LP_32_64_)
        U16     filler1;        /* For User32/Driver64 compatibility */
#endif  /* __LP64__ || _LP_32_64_ */

Insert the following after "U32 buffer_type;" of t_block_t typedef:
#if defined(__LP64__) || defined(_LP_32_64_)
        U32     filler2;        /* For User32/Driver64 compatibility */
#endif  /* __LP64__ || _LP_32_64_ */

          S32      tc_user_id;    /* CSL->TCU communication */
                           /* CCITT */
          U32      mtp_opc;
          S32      dialogue_id;   /* TCU->CSL communication */
                            /* ANSI */
          S32      trans_id;      /* TCU->CSL communication */

          S32      fwdlnk;   /* used by TCAP to link t_block to
                                the Dialogue/Transaction ID Table */
          union
          {
                   tc_chp_t        chp;
                   tc_dhp_t        dhp;
                   tc_thp_t        thp;
        }tc_user;
         U32     buffer_type;    /* set to 'TBLK' at
                                    initialization time */
} t_block_t;
```

* `primitive_type` (input/output)
  Specifies the TCAP component's origin. The value `TC_REQUEST` (or `1`) indicates that the TCAP component is a request; the value `TC_INDICATION` (or `2`) indicates that the TCAP component is a response. These values refer to constants defined in the include file `$SINAP_HOME/Include/tblock.h`.

* `primitive_code` (input/output)

  Specifies a code for the primitive. (These codes refer to constants defined in the include file `$SINAP_HOME/Include/tblock.h`.) The input parameter to the user-defined function is the index of the `T_Block`.

  If you are using the CCITT/TTC/NTT/China variants of the SINAP/SS7 system, the following codes indicate the type of primitive received:

  ```
  TC_UNI                  TC_RESULT_L
  TC_BEGIN                TC_RESULT_NL
  TC_CONTINUE             TC_U_ERROR
  TC_END                  TC_L_CANCEL
  TC_U_ABORT              TC_U_CANCEL
  TC_P_ABORT              TC_R_REJECT
  TC_NOTICE               TC_L_REJECT
  TC_INVOKE               TC_U_REJECT
  ```

  For applications based on 1993 TCAP standards, the following types of messages allow inclusion of a dialogue portion:

  ```
  TC_BEGIN
  TC_CONTINUE
  TC_END
  TC_U_ABORT
  ```

  The applications must contain the programming logic to process MSUs that contain a dialogue portion. Applications based on 1988 TCAP standards are not required to contain this programming logic. See Chapter 3 for more information on handling MSUs that contain dialogue portions. For examples of how to code an application to process MSUs containing dialogue portions, see the sample programs, `dials.c` and `dialr.c`, in the directory, `$SINAP_HOME/Samples/ccitt`.

  If you are using the ANSI variant of the SINAP/SS7 system, the following codes specify the type of primitive received:

  ```
  TC_UNI                  TC_INVOKE_NL
  TC_QRY_W_PERM           TC_INVOKE_L
  TC_QRY_WO_PERM          TC_RESULT_NL
  TC_CONV_W_PERM          TC_RESULT_L
  TC_CONV_WO_PERM         TC_U_ERROR
  TC_RESPONSE             TC_L_CANCEL
  TC_NO_RESPONSE          TC_U_CANCEL
  TC_U_ABORT              TC_R_REJECT
  TC_P_ABORT              TC_L_REJECT
  TC_NOTICE               TC_U_REJECT
  ```

* `tc_user_id` (input/output)
  Specifies the TCAP user ID.

* `mtp_opc` (input/output)
  Specifies the originating point code of the node that initiated the message.

Depending on which variant of the SINAP/SS7 system you are using, only one of the following two fields is applicable. Select the one for your network variant and specify the appropriate information.

* `dialogue_id` (input/output)
  (CCITT/TTC/NTT/China) Specifies the dialogue ID for this dialogue.

* `trans_id` (input/output)
  (ANSI) Specifies the transaction ID for this transaction.

The following fields are required for all network variants:

* `fwdlnk` (output)
  This field is used by TCAP to link the `T_Block` to an entry in the transaction ID table. You should not modify this field.

* `chp` (input/output)
  Specifies the `tc_chp_t` structure that contains the component-handling primitive being used. (For more information about the `tc_chp_t` structure, see "The Component-Handling Primitive Structure (`tc_chp_t`)" later in this section.)

Depending on which variant of the SINAP/SS7 system you are using, only one of the following two fields is applicable. Select the one for your network variant and specify the appropriate information.

* `dhp` (input/output)
  (CCITT/TTC/NTT/China) Specifies a `tc_dhp_t` structure that contains dialogue-handling information for the TCAP component. (For more information about the `tc_dhp_t` structure, see "The Dialogue-Handling Primitive Structure (`tc_dhp_t`)" later in this section.)

* `thp` (input/output)
  (ANSI) Specifies a `tc_thp_t` structure that contains transaction-handling information for the TCAP component. (For more information about the `tc_thp_t` structure, see "The Transaction-Handling Primitive Structure (`tc_thp_t`)" later in this section.)

The following field is required for all network variants. It is internal to the SINAP/SS7 system and should not be modified.

* `buffer_type` (output)

The Component-Handling Primitive Structure (`tc_chp_t`)

The `tc_chp_t` structure, which is used for all SINAP/SS7 variants, defines component-handling information. This structure is defined in the `tblock.h` include file and has the following format.

```
typedef struct tc_chp_s
{
    S16  linked_id;
    S16     corr_id;
    S16     invoke_id;
    U8      invoke_id_ind;
    U8      comp_type;
    U32  timer_value;
    U8      oper_class;
    U8      problem_type;
    U8      problem_code;
    U8      problem_specifier;
    BOOL    last_comp_ind;
    U16     extnd_data_size; /* contains data size in extended buffer */
    U8      dummy[1];
    U8      tot_data_len;
    U8      *extnd_data_ptr;/* When registered for XUDT/XUDTS */
                            /* points to extended data buffer */
#ifdef _LP_32_64_
    U32     filler;         /* For User32/Driver64 compatibility */
#endif  /* _LP_32_64_ */
    U8      data[MAX_DATA_SIZE_C]; /* data based on the primitive code*/
} tc_chp_t;
```

* `linked_id` (input/output)
  Links an operation invocation to a previous operation invocation.

* `corr_id` (input/output)
  Correlates an operation invocation to a previous operation invocation. The value of this field is in the range `0` to `255`. A value of `-1` indicates that the field is not applicable.

* `invoke_id` (input/output)
  Identifies an operation invocation. This field is used to indicate the MSU to which this TCAP component belongs. The value of this field is in the range `0` to `255`. A value of `-1` indicates that the field is not applicable.

* `invoke_id_ind` (output)
  Indicates whether `invoke_id` is valid. If this field is reset, `invoke_id` is valid; otherwise, it is not. A value of `INVALID_INVOKE_ID` indicates an invalid `invoke_id`.

* `comp_type` (output)
  Specifies the component type. A value of `HAND_OVER_COMP` indicates that the component is a handover component.

* `timer_value` (input/output)
  Specifies the maximum lifetime of `invoke_id`. This field is applicable only when the value of `primitive_code` is `TC_INVOKE` (CCITT), `TC_INVOKE_L` (ANSI), or `TC_INVOKE_NL` (ANSI). The value of this timer should be less than the setting of the environment variable `TCAP_TQ_BINS` or the default value of half the sum of `MIN_TQ_BINS` and `MAX_TQ_BINS` which are defined in `tcglob.h`. For example: (4 + 3601)/2 = 1802.

* `oper_class` (input/output)
  Identifies the type of operating class. Valid values, which refer to constants defined in `tblock.h`, are as follows:

  1 – Report both success and failure
  2 – Report only failure
  3 – Report only success
  4 – Report neither success nor failure

  The TC user should specify `OPER_CLASS_1` (report both success and failure) while preparing a component for a TCAP message to be sent to TCAP, unless the application specifically requires the use of Class 2, 3, or 4.

* `problem_type` (output)
  Indicates a problem-type tags. This field is valid only if the value of `primitive_code` is `TC_U_REJECT` or `TC_L_REJECT`. (See `tblock.h` for a list of valid problem-type tags).

* `problem_code` (output)
  Indicates the problem code. This field is valid only if the value of `primitive_code` is `TC_U_REJECT` or `TC_L_REJECT`. (For a list of valid problem codes, see the SINAP/SS7 `tblock.h` include file.)

* `problem_specifier` (output)
  Indicates the problem specifier. This field is valid only if the value of `primitive_code` is `TC_U_REJECT` or `TC_L_REJECT`.

* `last_comp_ind` (output)
  TCAP sets this field to indicate whether this is the last TCAP component of the MSU. This field is applicable only if the value of `primitive_code` is `TC_INVOKE` (CCITT), `TC_INVOKE_L` (ANSI), or `TC_INVOKE_NL` (ANSI).

* `*extnd_data_ptr` (output)
  When the application is registered at the TCAPX boundary, this field points to the extended data buffer (XUDT and XUDTS only).

* `extnd_data_size` (output)
  When the application is registered at the TCAPX boundary, this parameter contains the size of the data in the extended data buffer (XUDT/XUDTS only).

* `dummy` (output)
  This field is used as filler.

* `tot_data_len` (input/output)
  Specifies the total length of data in the `data` field.

* `data[MAX_DATA_SIZE]` (input/output)
  Provides the user data for the TCAP component, based on the value of `primitive_code`. As defined in the SINAP/SS7 `tblock.h` include file, this field provides up to 240 bytes of data for CCITT or 238 bytes of data for ANSI. For the CCITT/TTC/NTT/China variants of the SINAP/SS7 system, this field is formatted according to ITU-T (CCITT) Q.773 Recommendations. For the ANSI variant of the SINAP/SS7 system, this field is formatted according to ANSI T1.114.3 and T1.114.5 Recommendations. (`MAX_DATA_SIZE` is defined in the SINAP/SS7 `tblock.h` include file.)

Dialogue-Handling Primitive Structure (`tc_dhp_t`)

The `tc_dhp_t` structure, used for the CCITT, TTC, and China variants of the SINAP/SS7 system, defines dialogue-handling information. This structure is defined in the `tblock.h` include file and has the following format. See the ITU-T (CCITT) Q.771 Recommendations for a description of the non-SINAP-specific structure's fields.

```
typedef struc tc_dhp_s
{
           S16      msu_lost_count;         /* # of MSU lost by the driver */
           S16      write_free_mblk_count;  /* # of free M_Block, write queue */
           S16      read_free_mblk_count;   /* # of free M_Block, read queue */
           S16      read_queue_mblk_count;  /* # of M_Block in the read queue
                                            pending read */
/*****************************************************************************/
/* SCCP Called (destination address)/ Calling (origination address)*/
/* Party Address should be formatted as per the CCITT - Q713 */
/*****************************************************************************/

           U8       dest_addr[MAX_ADDR_LEN];/* destination TC-user */
           U8       orig_addr[MAX_ADDR_LEN];/* origination TC-user */
           U8       sccp_3rd_party_addr[MAX_ADDR_LEN];
           BOOL     comp_present_ind;       /* used in primitives of indication */
                                            /* type only */
                                            /* This field is set by the TCAP to */
                                            /* indicate that no component exist */
                    ;                       /* for the dialogue */
           U32      alt_DPC                 /* alternate DPC for routing label */
           U8       tb_options;             /* bit-masked tblock options */
#define    USE_ALT_DPC             0x01     /* use alt_DPC in MTP routing label */
                                            /* else use DPC from dest_addr */
#define    USE_DEST_ORIG_ADDR      0x02     /* use dest-addr and orig_addr in */
                                            /* SCCP header of the TCAP message */
#define    use_alt_DPC      tb_options     /* backward compatibility */
           U8       qlty_of_svc;            /* acceptable quality of service */
           U8       dialogue_end_type;

/* if dialogue is aborted by the transaction Sub Layer (Local or Remote),
then following field contain P Abort Cause or for TC_NOTICE, it contains
information indicating the reason for the exception report, for example that
the message was returned by the SCCP with the reason as specified in Q.711. */

           U8       pa_report_cause;        /* P Abort Cause or Report Cause */
/* if dialogue is aborted by TC-user, then following field contains
cause of abort and diagnostic information */
           U8       tot_ua_info_len;        /* user abort info length */
#define    MAX_UA_INFO_LEN_C       200
           U8       ua_info[MAX_UA_INFO_LEN_C];   /* user abort information */
           tc_association_t          ahp; ?* association handling part */
           U8       priority;               /* 0-3 priority values */
           U8       hop_count;              /* Hop count value to be inserted */
                                            /* into Extended Unit data */
                                            /* 0 = use default hop counter */
                                            /* >0 = insert this value in the
                                                hop counter parameter */
           U8       seq_control;            /* 0-31 slc values */
           U8       importance_parm;        /* ss7-2392: 1996 ITU-T Q.713 3.19 */
                                            /* The MSB is a flag to indicate if */
                                            /* SCCP Importance parm exists, if */
                                            /* it does, the 3 LSBs represents */
                                            /* the Importance values 0 to 7. */
} tc_dhp_t;
```

* `msu_lost_count` (output)
  Indicates the number of MSUs the driver has lost. This field is not used for sending TCAP components. The user application should initially set this to zero.

* `write_free_mblk_count` (output)
  Indicates the number of free `M_Blocks` in the write queue. This field is not used for sending TCAP components.

* `read_free_mblk_count` (output)
  Indicates the number of free `M_Block`s in the read queue. This field is not used for sending TCAP components.

* `read_queue_mblk_count` (output)
  Indicates the number of `M_Blocks` in the read queue that are awaiting a read operation. This field is not used for sending TCAP components.

* `dest_addr[MAX_ADDR_LEN]` (output)
  Indicates the SCCP called-party address (the address of the destination TCAP user), which is formatted according to ITU-T (CCITT) Q.713 Recommendations. This field is applicable if the value of `primitive_code` is `TC_UNI` or `TC_BEGIN`. (`MAX_ADDR_LEN` is defined in the `tblock.h` include file.)

* `orig_addr[MAX_ADDR_LEN]` (output)
  Indicates the SCCP calling-party address (the address of the originating TCAP user), which is formatted according to ITU-T (CCITT) Q.713 Recommendations. This field is applicable if the value of `primitive_code` is `TC_UNI` or `TC_BEGIN`. (`MAX_ADDR_LEN` is defined in the `tblock.h` include file.)

* `sccp_3rd_party_addr[MAX_ADDR_LEN]` (output)
  For a TCP/IP agent (registered at the SCCP boundary) receiving messages from a TCAP application, this field specifies the original calling party address information. The TCP/IP agent overwrites the original SCCP called party address information with its own point code and pseudo SSN to establish a two-way dialogue with an application registered at the TCAP boundary on the same SINAP node and system. In this case, the TCP/IP agent requires the original SCCP calling party address to correctly format and route messages back to the originating node over TCP/IP.

  For a TCAP application (accessed through the TCP/IP agent) originating a dialogue (for CCITT variants) or transaction (for ANSI variants), the field specifies the SCCP called party address of the TCAP application. In this case, the called party address is required because the original called party address provided in the tblock and mblock is configured to address the own signaling point (OSP) code and pseudo SSN of the TCP/IP agent running on the same SINAP node.

  The CASL transparently copies the `sccp_3rd_party_addr` field between the `tblock` and `mblock` in both directions when sending and receiving `tblock`s. The

SINAP driver initializes this field in the `mblock` to zeros when the SINAP node receives messages from Level 2 of the SS7 network.

The constant specified in the `MAX_ADDR_LEN` parameter is defined in the include files `$SINAP_HOME/Include/mblock.h.` and `$SINAP_HOME/Include/tblock.h.`

* `comp_present_ind` (output)
  TCAP sets this field to indicate whether a component exists for the dialogue. This field is used only for `INDICATION` type primitives.

* `alt_DPC` (output)
  Indicates the alternate DPC that is used in the MTP routing label in place of the DPC of the destination address.

* `tb_options` (input)
  Indicates the bit-masked `tblock` options. See `tblock.h` for values of this field, such as `USE_ALT_DPC`.

* `qlty_of_svc` (output)
  Indicates the protocol class of service for handling this primitive and the return-on-error indicator. This field is applicable only if the value of `primitive_code` is `TC_UNI` or `TC_BEGIN`. Valid values are as follows:

Value	Description
`CONN_LESS_SVC_CLASS_0(0)`	Connectionless Class 0, no return on error
`CONN_LESS_SVC_CLASS_1(1)`	Connectionless Class 1, no return on error
`0x80`	Connectionless Class 0, return on error
`0x81`	Connectionless Class 1, return on error

* `dialogue_end_type` (output)
  Indicates the way the dialogue is to end: `PREARRANGED_END (1)` indicates a prearranged end and `BASIC_END (2)` indicates a basic end. This field is applicable only if the value of `primitive_code` is `TC_END`.

* `pa_report_cause` (output)
  Indicates the `P_ABORT` or `REPORT` cause. See `tblock.h` for values of this field, such as `TSL_PA_TIMEOUT`.

* `tot_ua_info_len` (output)
  Indicates the length of the `ua_info` field. This field is applicable only if the value of `primitive_code` is `TC_U_ABORT`.

* `ua_info[MAX_UA_INFO_LEN_C]` (output)
  Indicates the reason for the abort; this field is applicable only if the value of `primitive_code` is `TC_U_ABORT`. If the dialogue was aborted by the TCAP user, this field contains the cause of the abort along with diagnostic information. (`MAX_UA_INFO_LEN` is defined in the `tblock.h` include file.)

* `ahp` (output)
  Specifies the `tc_association_t` structure that contains the application-context information for the MSU.

* `priority` (output)
  Indicates the message priority for the MSU. This parameter is valid only for SCCP Class 0 and Class 1 messages. The priority value can be in the range of 0 through 3 (lowest to highest).

* `hop_count` (output)
  Specifies the hop count value to be inserted into the MSU. (XUDT only) The mandatory hop counter limits the number of global title translations (GTTs) that can be performed on the message. If the `hop_count` value is less than 1 or greater than 15, the SINAP/SS7 software defaults to the value 10. Any `hop_count` value between 1 and 15 is inserted into the MSU.

* `seq_control` (output)
  Indicates the value to use for the signaling link selection (SLS) field of the MSU's MTP routing label. This parameter is valid for SCCP Protocol Class 1 messages only. The valid value range for the TTC variant is 0 through 15. For all other variants excluding ANSI, the valid value range is 0 through 31.

  For the ANSI network variant, `seq_control` can have a value in the range of `0` through `255` if the SINAP user specified an eight-bit SLS through the `CHANGE-SLSTYPE` MML command. In all other cases the `seq_control` field can have a value in the range `0-31` (a five-bit SLS). See "SINAP/SS7 Interaction with the SS7 Network" in Chapter 2 for more information.

* `importance_parm` (input/output)
  For the CCITT (ITU-T) variant, if the environment variable SCCP_ITU96_IMPORTANCE_PARM is set, and the user registers at the TCAPX boundary, this field holds the importance parameter (ss7-2392: 1996 ITU-T Q.713 3.19). The MSB is used as a bit flag to indicate if the SCCP optional Importance parameter is included in the SCCP XUDT/XUDTS message. If it is, then the 3 LSBs represent Importance values 0 to 7.

The `tc_association_t` Structure

The `tc_association_t` structure contains the dialogue portion of the MSU. The dialogue portion defines the application-context name and optional user information to be used for the

application-context dialogue. The format of the `tc_association_t` structure is defined in the include file `$SINAP_HOME/Include/tblock.h` and has the following format.

```
typedef struct assoc
{
             int                 dlgInfoPresent;
             int                 pduType;
               acn_t               *applicationContextName;
#ifdef _LP_32_64_
            U32 filler; /* For User32/Driver64 compatibility */
#endif  /* _LP_32_64_ */
              tc_user_data_t      userInformation;
             int                 abortSource;
             int                 result;
             int                 resultSourceDiag;
               int                 resultSourceDiagValue;
}tc_association_t;
```

* `dlgInfoPresent` (input)
  The TC user initializes this field to one of the following values to indicate whether the TCAP component contains a dialogue portion:

  - `TRUE` (`1`) indicates the presence of a dialogue portion.
  - `FALSE` (`0`) indicates that no dialogue portion is present.

* `pduType` (output)
  TCAP initializes this field to indicate the type of ACSE APDU present in the dialogue portion of the TCAP component. Possible values are `AARQ`, `AUDT`, `ABRT`, and `AARE`; see `tcap.h`.

* `applicationContextName` (input)
  The TC user initializes this field to the name of an `acn_t` structure that contains the application-context name for the dialogue. For more information, see "The acn_t Structure" later in this chapter.

* `userInformation` (input)
  The TC user initializes this field to the name of a `tc_user_data_t` structure that contains optional user information for the dialogue. For more information, see "The tc_user_data_t Structure" later in this chapter.

* `abortSource` (output)
  When an error occurs, TCAP initializes this field to one of the following values to indicate who aborted the dialogue:

- `dialogue-service-provider` (`0`) indicates that the service provider aborted the dialogue, possibly due to a syntax error.
- `dialogue-service-user` (`1`) indicates that the TC user aborted the dialogue, possibly because the specified application-context name is not supported.

* `result` (output)
  TCAP initializes this field to one of the following values to indicate the status of the association request:

  - `accepted` (`0`) indicates that the association request has been accepted.
  - `reject-permanent` (`1`) indicates that the association request has been denied.

* `resultSourceDiag` (output)
  When an error occurs, TCAP initializes this field to a particular type of source diagnostic. Possible values are `TC_SERVICE_USER` and `TC_SERVICE_PROVIDER`.

* `resultSourceDiagValue` (input)
  When aborting a request, the service provider initializes this field to one of the following values:

  - `NULL` (`0`)
  - `no-reason-given` (`1`) can indicate a syntax error
  - `no-common-dialogue-portion` (`2`)

  When aborting a request, the TC user initializes this field to one of the following values:

  - `NULL` (`0`)
  - `no-reason-given` (`1`)
  - `application-context-name-not-supported` (`2`)

The acn_t Structure

The acn_t structure contains the application-context name for the application-context dialogue. The acn_t structure is defined in the include file $SINAP_HOME/Include/tblock.h and has the following format.

```
typedef struct
{
          long length;
          union {
          char   *long_buf;
          char   short_buf[MAX_ACN_SIZE];
          } b;
} acn_t;
```

When you design an application to initiate an application-context dialogue or respond to a TC_BEGIN message that contains a dialogue portion, make sure the application initializes the acn_t structure to the appropriate application-context name by using either of the following methods:

- The application can call the function, tc_objmk(), to create the OID, then cast the results to an acn_t structure pointer, as shown in the following example. (Note that the acn_t structure stores the object identifier.)

```
ptblk->tc_user.dhp.ahp.applicationContextName =
     (acn_t *) tc_objmk(0x02, 0x04, 0x01, 0x01, 0x08,
     0x03, END_OF_OID);
```

NOTE

objmk() is a utility supplied with the ASN.1 compiler.

- The application can initialize the fields of the acn_t structure to the application-context name. Note that you **must** define the application-context name as a properly formatted ASE OID, encoded according to the rules documented in ITU-T Recommendation X.690, *Basic Encoding Rules*.

The `tc_user_data_t` Structure

The `tc_user_data_t` structure contains optional user information for the application-context dialogue (such as a password, application-initialization data, protocol-version information, and so on). This user information is exchanged independently of the remote-service operation and is transparent to TCAP. The `tc_user_data_t` structure is defined in the `$SINAP_HOME/Include/tblock.h` include file and has the following format.

```
typedef struct tc_user_data_s
{
          int       size;
#if defined(__LP64__) || defined(_LP_32_64_)
          U32       filler; /* For User32/Driver64 compatibility */
#endif  /* __LP64__ || _LP_32_64_ */
          char      userInfo[MAX_USER_INFO];
}tc_user_data_t;
```

The fields of the `tc_user_data_t` structure are as follows:

* `size`
  The TC user initializes this field to the length of the `userInfo` field.

* `userInfo[MAX_USER_INFO]`
  The TC user initializes this field to define optional user information for the application-context dialogue. Format this field according to Section 4.2.3 of the 1993 edition of ITU-T Recommendation Q.773. According to Table 49 of the recommendation, this field must be defined as an ASN.1-encoded sequence of externals. (`MAX_USER_INFO` is defined in the `$SINAP_HOME/Include/tblock.h` include file.)

  You can use the ASN.1 compiler to properly format the value of this field. The compiler creates an ASN.1-encoded sequence of externals from the user information that you provide. You can then write the compilation results to the `userInfo` field.

The Transaction-Handling Primitive Structure (`tc_thp_t`)

The `tc_thp_t` structure, which is used for the ANSI variant of the SINAP/SS7 system, defines transaction-handling information. This structure is defined in the `tblock.h` include file and has the following format. See the ANSI T1.114.1 Recommendations for generic descriptions of this structure's fields.

```
typedef struct tc_thp_s
{
          S16      msu_lost_count;
          S16      write_free_mblk_count;
          S16      read_free_mblk_count;
          S16      read_queue_mblk_count;
          U8       dest_addr[MAX_ADDR_LEN];
          U8       orig_addr[MAX_ADDR_LEN];
          U8       sccp_3rd_party_addr[MAX_ADDR_LEN];
          U8       dest_tid[MAX_TID_SIZE];
          U8       orig_tid[MAX_TID_SIZE];
          U8       orig_tid_len;
          U8       local_tid[MAX_TID_SIZE];
          U8       local_tid_len;
          U8       packet_type;
          U8       qlty_of_svc;
          U8       seq_control;
          U8       hop_count;
          U8       priority;
          BOOL     comp_present_ind;
          U8       tb_options;
          U8       alt_DPC[DPC_LEN];
          U8       fictitious_OPC;
          U8       trans_end_type;
          U8       pa_report_cause;
          U8       tot_ua_info_len;
          U8       ua_info[MAX_UA_INFO_LEN];
} tc_thp_t;
```

* `msu_lost_count` (input/output)
  Indicates the number of MSUs the SINAP driver has lost. The user application should set this field to zero initially.

* `write_free_mblk_count` (output)
  Indicates the number of free `M_Blocks` in the SINAP driver write queue.

* `read_free_mblk_count` (output)
  Indicates the number of free `M_Blocks` in the read queue.

* `read_queue_mblk_count` (output)
  Indicates the number of `M_Blocks` in the read queue that are awaiting a SINAP driver read operation.

* `dest_addr[MAX_ADDR_LEN]` (input/output)
  Indicates the SCCP called-party address (the address of the destination TCAP user), which is formatted according to ANSI T1.113.3 Recommendations. This field is applicable only if the value of `primitive_code` is `TC_UNI`, `TC_QRY_W_PERM`, or `TC_QRY_WO_PERM`. (`MAX_ADDR_LEN` is defined in the SINAP/SS7 `tblock.h` include file.)

* `orig_addr[MAX_ADDR_LEN]` (input/output)
  Indicates the SCCP calling-party address (the address of the origination TCAP user), which is formatted according to ANSI T1.113.3 Recommendations. This field is applicable only if the value of the `primitive_code` parameter is `TC_UNI`, `TC_QRY_W_PERM`, or `TC_QRY_WO_PERM`. (`MAX_ADDR_LEN` is defined in the SINAP/SS7 `tblock.h` include file.)

* `sccp_3rd_party_addr[MAX_ADDR_LEN]` (input/output)
  For a TCP/IP agent (registered at the SCCP boundary) receiving messages from a TCAP application, this field specifies the original calling party address information. The TCP/IP agent overwrites the original SCCP called party address information with its own point code and pseudo SSN to establish a two-way dialogue with an application registered at the TCAP boundary on the same SINAP node and system. In this case, the TCP/IP agent requires the original SCCP calling party address to correctly format and route messages back to the originating node over TCP/IP.

  For a TCAP application (accessed through the TCP/IP agent) originating a dialogue (for CCITT variants) or transaction (for ANSI variants), the field specifies the SCCP called party address of the TCAP application. In this case, the called party address is required because the original called party address provided in the tblock and mblock is configured to address the own signaling point (OSP) code and pseudo SSN of the TCP/IP agent running on the same SINAP node.

  The CASL transparently copies the `sccp_3rd_party_addr` field between the `tblock` and `mblock` in both directions when sending and receiving `tblock`s. The SINAP driver initializes this field in the `mblock` to zeros when the SINAP node receives messages from Level 2 of the SS7 network.

  The constant specified in the `MAX_ADDR_LEN` parameter is defined in the include files `$SINAP_HOME/Include/mblock.h.` and `$SINAP_HOME/Include/tblock.h.`

* `dest_tid[MAX_TID_SIZE]` (input/output)
  Indicates the destination transaction ID. For a particular transaction ID, the TC user should save the destination address, the origination address, and this value. (`MAX_TID_SIZE` is defined in the SINAP/SS7 `tblock.h` include file.)

* `orig_tid[MAX_TID_SIZE]` (input/output)
  Indicates the origination transaction ID. (`MAX_TID_SIZE` is defined in the SINAP/SS7 `tblock.h` include file.)

* `orig_tid_len` (input/output)
  Indicates the length of the origination transaction ID. Valid values are `0` or `4`.

* `local_tid[MAX_TID_SIZE]` (output)
  Indicates the local transaction ID. (`MAX_TID_SIZE` is defined in the SINAP/SS7 `tblock.h` include file.)

* `local_tid_len` (output)
  Indicates the length of the local transaction ID. Valid values are `0` or `4`.

* `packet_type` (output)
  Indicates the TCAP packet type. A value of zero indicates that TCAP will translate the TC primitive to a TCAP packet type. A non-zero value indicates that the SINAP/SS7 system will use this field directly as a TCAP packet. See `tcap.h` for a list of nonzero packet types, such as `TSL_PT_UNI`.

* `qlty_of_svc` (input/output)
  Indicates the protocol class of service to use when sending this MSU and the return-on-error indicator. This field is applicable only if the value of `primitive_code` is `TC_UNI` or `TC_BEGIN`. Valid values are as follows:

Value	Description
`CONN_LESS_SVC_CLASS_0(0)`	Connectionless Class 0, no return on error
`CONN_LESS_SVC_CLASS_1(1)`	Connectionless Class 1, no return on error
`0x80`	Connectionless Class 0, return on error
`0x81`	Connectionless Class 1, return on error

* `seq_control` (input/output)
  Indicates the signaling link selection (SLS) code of the link over which messages are sent.

  For quality of service requiring Class 0 (`CONN_LESS_SVC_CLASS_0`), set the `seq_control` field to 0. For quality of service requiring Class 1 (`CONN_LESS_SVC_CLASS_1`), set the `seq_control` field to the appropriate service link selection (SLS) value (0 through 31).

* `hop_count` (output)
  Specifies the hop count value to be inserted into the MSU. (XUDT only) The mandatory hop counter limits the number of global title translations (GTTs) that can be performed on the message. The valid range of values for the hop count is 1 through 15. The default count is 12.

* `priority` (input/output)
  Indicates the transaction priority (in the range `0` to `3`) for each transaction-handling primitive. The priority of the MSU's SIO octet is based on this value.

* `tb_options` (input)
  Indicates one of the following bit-masked `tblock` option codes to use for message routing.

Code	Description
0x01	Use the `alt_DPC` in the MTP routing label if set. Otherwise, use the DPC specified in the `dest_addr` field.
0x02	Use the destination address specified in the `dest_addr` field and the origination address specified in the `orig_addr` field of the SCCP header of the TCAP message. (Used for backward compatibility.)

* `comp_present_ind` (output)
  Indicates whether the transaction contains a component: `0` indicates that there is no component present; `1` indicates that there is a component present. This field is used for `INDICATION` type primitives only.

* `alt_DPC[DPC_LEN]` (output)
  Indicates the alternate DPC that is used in the MTP routing label in place of the DPC of the destination address. (`DPC_LEN` is defined in the SINAP/SS7 `tblock.h` include file.)

* `fictitious_OPC` (input/output)
  Indicates whether the MTP routing label contains the point code of the originating address or a fictitious OPC. A value of `1` indicates the MTP routing label contains a fictitious OPC. Otherwise, the MTP routing label uses the OPC in `orig_addr`.

* `trans_end_type` (input/output)
  Indicates how the transaction is to be ended: `PREARRANGED_END(1)` indicates a prearranged end and `BASIC_END(2)` indicates a basic end. This field is applicable if the value of `primitive_code` is `TC_QRY_W_PERM`.

* `pa_report_cause` (output)
  Indicates the `P_ABORT` or `REPORT` cause. If the user aborts the transaction, this field contains the cause of the abort, along with diagnostic information. See `tcap.h` for possible values of this field, such as `TSL_PA_TIMEOUT`.

* `tot_ua_info_len` (output)
  Indicates the length of the `ua_info` field. This field is applicable only if the value of `primitive_code` is `TC_U_ABORT`.

* `ua_info [MAX_UA_INFO_LEN_A]` (output)
  Indicates the reason for the abort. If the user aborts the transaction, this field contains the cause of the abort along with diagnostic information. This field is applicable only if the value of `primitive_code` is `TC_U_ABORT`.

FILES

`$SINAP_HOME/Include/arch.h`, `ca_error.h`

RETURN VALUES

The `ca_get_tc()` function returns an index to the next available `T_Block`. If the function returns `-1`, there is an error; see `errno` for error number and description. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

A possible CASL value for `errno` follows.

Value	Meaning
`CA_ERR_NO_MSUS`	There are no MSUs in the batch buffer.

The TCAP can return the following errors.

Error	Action
`TC_ERR_NOT_REG_AT_TCAP_BOUNDARY`	Reregister the application at the TCAP boundary using `ss7_input_boundary= SS7_INPUT_BOUNDARY_TCAP`. The SINAP/SS7 system provides an immediate return; `ca_get_tc()` is not executed.
`TC_ERR_OUT_OF_TRANS_ID`	Increase the value of the `max_trans_id` parameter of the `ca_register()` function to reregister the application with a greater number of transaction IDs. The SINAP/SS7 system deallocates the `T_Block`; `ca_get_tc()` is not executed. `TC_CONTINUE` (CCITT) or `TC_CONV_W_PERM` and `TC_CONV_WO_PERM` (ANSI); `TC_END` (CCITT) or `TC_RESPONSE` (ANSI); and `TC_P_ABORT` and `TC_U_ABORT` primitives do not return this error.

Error	Action
TC_ERR_T_BLOCK_CAPACITY_EXHAUSTED	Increase the value of the `tc_count` parameter of the `ca_register()` function to reregister the application with a greater number of `T_BLOCK`s. The SINAP/SS7 system deallocates the `T_Block`; `ca_get_tc()` is not executed.
TC_ERR_INV_TSL_STATE	Report this error, along with all debug information, to the Customer Assistance Center (CAC). The SINAP/SS7 system provides an immediate return; `ca_get_tc()` is not executed.
TC_ERR_INV_TSL_EVENT	Report this error, along with all debug information, to the CAC. SINAP/SS7 provides an immediate return; `ca_get_tc()` is not executed.
TC_ERR_INV_ISM_STATE	Report this error, along with all debug information, to the CAC. SINAP/SS7 provides an immediate return; `ca_get_tc()` is not executed.
TC_ERR_INV_ISM_EVENT	Report this error, along with all debug information, to the CAC. SINAP/SS7 provides an immediate return; `ca_get_tc()` is not executed.
TC_ERR_TSL_TEQ_OVERFLOW	Report this error, along with all debug information, to the CAC. SINAP/SS7 provides an immediate return; `ca_get_tc()` is not executed.
TC_ERR_ISM_TEQ_OVERFLOW	Report this error, along with all debug information, to the CAC. SINAP/SS7 provides an immediate return; `ca_get_tc()` is not executed.
TC_ERR_PTR_TO_USF_NOT_SET	No user-defined pointer.
TC_ERR_OUT_OF_TC_USER_ID	A user-supplied function has exhausted its supply of user IDs. Check to ensure that the supply of IDs is large enough.
TC_ERR_TRANS_ID_NOT_ASSIGNED	Before calling `ca_get_tc()`, the process must call the `ca_get_trans_id()` function to obtain a transaction ID for the transaction.

Error	Action
TC_ERR_TCAP_OWN_TRANS_ID	The transaction is not under application control. A TC_RESPONSE primitive will release this transaction. (As per ANSI Recommendations, a TC_NO_RESPONSE primitive, which is prearranged by both TC users, causes messages to be discarded rather than being sent to the network.) No further action is required.

The ca_get_tc() function calls the ca_alloc_tc() function and can also return the errors listed under that function. Under certain circumstances ca_get_tc() may call ca_dealloc_tc(); therefore, it may return the errors listed under ca_dealloc_tc().

SEE ALSO

ca_alloc_tc(), ca_dealloc_tc(), ca_process_tc(), ca_put_tc()

ca_get_tc_ref()

SYNOPSIS

```
int       ca_get_tc_ref(
            BOOL      *prefwait,
            int       (*pfunc)(S32));
{
}
```

DESCRIPTION

The `ca_get_tc_ref()` function returns an index to an incoming `T_Block`, which contains a TCAP component. The `ca_get_tc_ref()` function is almost identical to the `ca_get_tc()` function; however, in place of the `fwait` parameter (which is passed by value), `ca_get_tc_ref()` uses the parameter `*prefwait`.

The `*prefwait` parameter points to the global variable `REFWAIT`, whose value is a Boolean indicator that specifies whether the `ca_get_tc_ref()` function call is to execute in blocking or nonblocking mode: `1` specifies blocking mode and `0` specifies nonblocking mode. `REFWAIT` is defined in the include file `sinapintf.h`.

In blocking mode, `ca_get_tc_ref()` will not return until it reads a `T_Block` from the queue; if there are none, normal application processing is suspended until one arrives. In nonblocking mode, `ca_get_tc_ref()` returns an error message if there is nothing on the queue; normal application processing is not suspended as it is when the function is called in blocking mode.

Since `*prefwait` is a pointer to the variable `REFWAIT`, it is possible for the calling process, an interrupt-handler function, or another application process to dynamically change the execution mode of the `ca_get_tc_ref()` function call by changing the value of `REFWAIT` (for example, from blocking to nonblocking mode).

NOTE

The CASL function `ca_get_tc()` calls the `ca_get_tc_ref()` function and initializes `REFWAIT` to the value of its `fwait` parameter. The `ca_get_tc()` function also passes a pointer to `REFWAIT` to the `ca_get_tc_ref()` function's `*prefwait` parameter, thus enabling the calling

process, an interrupt-handler function, or another application process to dynamically change the execution mode of the `ca_get_tc()` function call.

PARAMETERS

* `*prefwait` (input)

  Specifies a pointer to the `REFWAIT` global variable, which is a Boolean indicator that defines whether to execute the `ca_get_tc_ref()` function call in blocking or nonblocking mode. Set `REFWAIT` to `1` to execute the `ca_get_tc_ref()` function call in blocking mode (no return until a `T_Block` is read from the queue); set `REFWAIT` to `0` for nonblocking mode (return an error if there is nothing on the queue).

* `*pfunc` (input)

  Specifies a pointer to a user-supplied function that returns a TCAP user dialogue/transaction ID. Specify `0` if you want TCAP to retrieve an index to the next available `T_Block`.

  To control the way TCAP user IDs are assigned to TCAP dialogues/transactions, use this parameter to specify a pointer to a user-supplied function that returns a unique TCAP user ID in the `tc_user_id` field of the `T_Block`.

FILES

`arch.h`, `ca_error.h`

RETURN VALUES

The `ca_get_tc_ref()` function returns an index to the next available `T_Block`. If the function returns `-1`, there is an error; see `errno` for the error number and meaning. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

A possible CASL value for `errno` is as follows:

Error	Meaning
`CA_ERR_NO_MSUS`	There are no MSUs on the queue.

The TCAP can return the following errors.

Error	Action
TC_ERR_NOT_REG_AT_TCAP_BOUNDARY	Re-register the application with the registration parameter ss7_input_boundary set to SS7_INPUT_BOUNDARY_TCAP. The SINAP/SS7 system provides an immediate return; ca_get_tc_ref() is not executed.
TC_ERR_OUT_OF_TRANS_ID	Re-register the application and increase the value of the max_dialogue_id registration parameter. The SINAP/SS7 system deallocates the T_Block and does not execute ca_get_tc_ref(). CONTINUE, END, and ABORT primitive code messages do not return this error.
TC_ERR_OUT_OF_DIAL_ID	Close open dialogues with END, release dialogue IDs, or allocate more dialogue IDs and re-register the application. If the primitive code is TC_BEGIN, the SINAP/SS7 system stops the transaction timer, releases the transaction ID, and deallocates the T_Block. CONTINUE, END, and ABORT primitive code messages do not return this error.
TC_ERR_TRANS_ID_ALREADY_RELEASED	Use a BEGIN message to initiate a dialogue in TCAP. The SINAP/SS7 system provides an immediate return; ca_get_tc_ref() is not executed.
TC_ERR_T_BLOCK_CAPACITY_EXHAUSTED	Re-register the application with a greater number of T_Blocks. The SINAP/SS7 system deallocates the T_Block and does not execute ca_get_tc_ref().

Error	Action
TC_ERR_INV_TSL_STATE	Report this error to the CAC, along with all relevant debug information. The SINAP/SS7 system provides an immediate return; `ca_get_tc_ref()` is not executed.
TC_ERR_INV_TSL_EVENT	Report this error to the CAC, along with all relevant debug information. The SINAP/SS7 system provides an immediate return; `ca_get_tc_ref()` is not executed.
TC_INV_ISM_STATE	Report this error to the CAC, along with all relevant debug information. The SINAP/SS7 system provides an immediate return; `ca_get_tc_ref()` is not executed.
TC_INV_ISM_EVENT	Report this error to the CAC, along with all relevant debug information. The SINAP/SS7 system provides an immediate return; `ca_get_tc_ref()` is not executed.
TC_ERR_INV_TEQM_OVERFLOW	Report this error to the CAC, along with all relevant debug information. The SINAP/SS7 system provides an immediate return; `ca_get_tc_ref()` is not executed.
TC_ERR_TSL_TEQ_OVERFLOW	Report this error to the CAC, along with all relevant debug information. The SINAP/SS7 system provides an immediate return; `ca_get_tc_ref()` is not executed.
TC_ERR_ISM_TEQ_OVERFLOW	Report this error to the CAC, along with all relevant debug information.
TC_ERR_PTR_TO_USF_NOT_SET	No user-defined pointer.

Error	Action
TC_ERR_OUT_OF_DIAL_ID	There is a problem with dialogue IDs. Either close open dialogues with END, release dialogue IDs, or allocate more dialogue IDs and re-register the application.
TC_ERR_OUT_OF_TC_USER_ID	A user-supplied function has exhausted its supply of user IDs. Make sure that the supply of IDs is large enough.
TC_ERR_DIAL_ID_ALREADY_RELEASED	The application must obtain a dialogue ID before starting the dialogue.
TC_ERR_TCAP_OWN_DIAL_ID	The dialogue is not under application control. An END message will release this dialogue. (As defined in ITU-T (CCITT) Recommendation Q.775 and ANSI Recommendation T1.114.5, a prearranged END will not cause messages to be sent to the network.) No further action is required.

ca_get_trans_id()

SYNOPSIS

```
int    ca_get_trans_id();
```

DESCRIPTION

The `ca_get_trans_id()` function returns a unique transaction ID for a TCAP transaction. The calling process uses this transaction ID to identify multiple TCAP components that belong to a single transaction. This function is used in the ANSI variant of the SINAP/SS7 system. The CCITT/TTC/NTT/China variants of the SINAP/SS7 system uses the function: `ca_get_dial_id()`.

This transaction ID remains in effect for the duration of the transaction. The calling process can call the `ca_rel_trans_id()` function to explicitly release the transaction ID, or it can do nothing. If the calling process does not call `ca_rel_trans_id()`, TCAP will release the transaction ID when it detects that the transaction has terminated. To effectively manage the supply and use of transaction IDs, the calling process should call `ca_rel_trans_id()` after terminating the transaction.

For the SINAP/SS7 system to route TCAP components belonging to a specific transaction, the TCAP and TCAP user must assign a transaction ID locally. ANSI Recommendations specify that the TCAP user must assign a transaction ID when it initiates a new transaction. To manage TCAP transactions, TCAP assigns internal transaction IDs (IDs that are visible only to SINAP/SS7 processes and the SINAP/SS7 TCAP user) whenever a remote TC-user issues a `TC_UNI`, `TC_QRY_W_PERM`, or `TC_QRY_WO_PERM` primitive.

Because the TCAP component sublayer uses the transaction ID as the key to access, store, and retrieve TCAP components, there is a potential problem when two different functions assign transaction IDs from the same pool of values. To accommodate the transaction ID assignment by both the TCAP user and TCAP, the SINAP/SS7 system provides two assignment paths: one for input and one for output.

On input, the TCAP user must issue a `ca_get_tc()` function call to receive the next available primitive (transaction or component). The `trans_id` parameter in the `ca_get_tc()` function contains the address of the function that the TCAP user wants TCAP to call when a remote node starts a new transaction; TCAP calls this function only when a new transaction is started.

On output, the TCAP user obtains a TCAP transaction ID by calling `ca_get_trans_id()`, then obtains a `T_Block` by calling `ca_alloc_tc()`. The TCAP user creates the `TC_UNI`, `TC_QRY_W_PERM`, or `TC_QRY_WO_PERM` primitive in the `T_Block` and inserts the TCAP user and TCAP IDs in the `T_Block` `trans_id` parameter. When `ca_put_tc()` is called, the TCAP and TCAP user are aware of both parts of the transaction ID. All subsequent calls to `ca_put_tc()` must refer to `T_Blocks` that contain this transaction ID. For debugging, the TCAP user should also place its transaction ID in the field allocated for that purpose.

FILES

`arch.h`, `ca_error.h`, `tblock.h`

RETURN VALUES

The `ca_get_trans_id()` function returns a unique transaction ID. If the function returns `-1`, there is an error; see `errno` for error number and description. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

The TCAP can return the following error.

Error	Action
`TC_ERR_OUT_OF_TRANS_ID`	Do one of the following: • Issue a `TC_RESPONSE` primitive. • Call the `ca_rel_trans_id()` function to release transaction IDs that are no longer being used. • Call the `ca_register()` function and increase the value of the `max_trans_id` parameter.

SEE ALSO

`ca_get_tc()`, `ca_rel_trans_id()`

ca_process_tc()

SYNOPSIS

```
int     ca_process_tc(
          proc_tc_t    *pstev,
           int          (*pfunc)(S32));
```

DESCRIPTION

An application process calls `ca_process_tc()` to take control of TCAP component processing. This function simplifies service creation by accessing inbound TCAP components and distributing them to client-specified processing functions that are based on a finite state machine operation model. In this manner, the `ca_process_tc()` function acts as a main program for a server process and is thus suitable for cloned operation.

To use `ca_process_tc()`, you must construct a state/event table that describes how the function is to process components. The table must take the format of the structure `proc_tc_t`, which is defined in the include file `proc_tc.h`. You must also construct a series of functions that process individual TCAP components under specific constraints. After you create these items, you can create the main program of the client application, using the basic flow shown here.

```
main()
{
ca_register();
ca_process_tc();
ca_terminate();
}
```

PARAMETERS

* `pstev` (input)
  Specifies a pointer to a `proc_tc_t` structure that contains the address of each function for each state and event listed in a state/event table you provide. For more information, see "The `proc_tc_t` Structure" later in this section.

* `pfunc` (input)
  Specifies a pointer to a user-supplied function.

The `proc_tc_t` Structure

To construct a state/event table, use the `proc_tc_t` structure, which is shown here. The structure is defined in the include file `proc_tc.h`.

```
typedef struct proc_tc_s
{
      entry_t entry[MAX_EVENTS];
} proc_tc_t;
```

* `entry[MAX_EVENTS]` (input)
  Specifies the next entry to process. For more information, see "The `entry_t` Structure," which follows. (`MAX_EVENTS` is defined in the SINAP/SS7 `proc_tc.h` include file.)

  NOTE

  You should declare the state/event table as follows:

  ```
  proc_tc_t tc_state_tbl[MAX_STATES];
  ```

The `entry_t` Structure

The `entry_t` structure contains the following fields and is defined in the include file `proc_tc.h`.

```
typedef struct entry_s
{
        U8     event_type;
        int    event;
        void   (*pfunction)();
} entry_t;
```

* `event_type` (input)
  Specifies the type of event to process. Possible values are `1` for IPC events and `2` for TCAP events.

* `event` (input)
  Specifies the event to process. See the include file `iblock.h` for a list of possible IPC events and messages. See the include file `tblock.h` for a list of TCAP primitive types.

* `pfunction` (input)
  Specifies a pointer to a user-supplied function that performs state-event processing.

FILES

`arch.h`, `ca_error.h`, `iblock.h`, `tblock.h`, `proc_tc.h`

RETURN VALUES

The `ca_process_tc()` function returns `-1` if there is an error; see `errno` for error number and description. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

The TCAP can return the following error.

Error	Action
`TC_ERR_NOT_REG_AT_TCAP_BOUNDARY`	Reregister the application at the TCAP boundary using `ss7_input_boundary= SS7_INPUT_BOUNDARY_TCAP`. The SINAP/SS7 system provides an immediate return; `ca_process_tc()` is not executed.

This function performs a `ca_get_msg()` and can also return the errors listed under that function.

SEE ALSO

`ca_alloc_tc()`, `ca_dealloc_tc()`, `ca_get_msg()`, `ca_get_tc()`, `ca_get_trans_id()`, `ca_put_tc()`, `ca_rel_trans_id()`

ca_put_tc()

SYNOPSIS

```
int     ca_put_tc(
          S32     tb_index);
```

DESCRIPTION

The TCAP client application calls the `ca_put_tc()` function to deliver a TCAP component to another TCAP application. When this function is called, TCAP packages the TCAP component in an MSU and calls the `ca_put_msu_int()` function to deliver the MSU to the SCCP. The SCCP then takes over processing, calling the `ca_put_msu()` function to deliver the MSU to the MTP for transmission to its destination.

PARAMETERS

* `tb_index` (input)

  Specifies the index to the `T_Block` array returned by the client's primitive request. The structure of the `T_Block` is defined in the include file `tblock.h`. For a description of this structure's fields, see the following section, "The `T_Block` Structure (`t_block_t`)."

The T_Block Structure (t_block_t)

For the ca_put_tc() function to work, you must set the following fields in t_block_t structure, which is defined in the include file tblock.h.

```
 typedef struct t_block_s
{
          U8       primitive_type;
          U8      primitive_code;
#if defined(__LP64__) || defined(_LP_32_64_)
        U16     filler1;        /* For User32/Driver64 compatibility */
#endif  /* __LP64__ || _LP_32_64_ */

Insert the following after "U32 buffer_type;" of t_block_t typedef:
#if defined(__LP64__) || defined(_LP_32_64_)
        U32     filler2;        /* For User32/Driver64 compatibility */
#endif  /* __LP64__ || _LP_32_64_ */

          S32      tc_user_id;    /* CSL->TCU communication */
                           /* CCITT */
          U32      mtp_opc;
          S32      dialogue_id;   /* TCU->CSL communication */
                           /* ANSI */
          S32      trans_id;      /* TCU->CSL communication */

          S32      fwdlnk;   /* used by TCAP to link t_block to
                                the Dialogue/Transaction ID Table */
          union
          {
                   tc_chp_t        chp;
                   tc_dhp_t        dhp;
                   tc_thp_t        thp;
        }tc_user;
         U32     buffer_type;    /* set to 'TBLK' at
                                    initialization time */
} t_block_t;
```

* primitive_type (input/output)

  Specifies the TCAP component's origin as shown in the following chart.

Value	Origin
TC_REQUEST (1)	Request
TC_INDICATION (2)	Response

Value	Origin
TC_REQUESTX (3)	Request. Ensures components are carried in an XUDT message. This primitive is valid only if the application is registered with CASL at the input boundary SS7_INPUT_BOUNDARY_TCAPX.

These values refer to constants defined in the include file `$SINAP_HOME/Include/tblock.h`.

* `primitive_code` (input/output)
  Specifies a code for the primitive. (These codes refer to constants defined in the include file `$SINAP_HOME/Include/tblock.h`.) The input parameter to the user-defined function is the index of the `T_Block`.

  If you are using the CCITT/TTC/NTT/China variants of the SINAP/SS7 system, specify one of the following codes to indicate the type of primitive being used:

```
TC_UNI                  TC_RESULT_L
TC_BEGIN                TC_RESULT_NL
TC_CONTINUE             TC_U_ERROR
TC_END                  TC_L_CANCEL
TC_U_ABORT              TC_U_CANCEL
TC_P_ABORT              TC_R_REJECT
TC_NOTICE               TC_L_REJECT
TC_INVOKE               TC_U_REJECT
```

  If you are using the ANSI variant of the SINAP/SS7 system, specify one of the following codes to indicate the type of primitive being used:

```
TC_UNI                  TC_INVOKE_NL
TC_QRY_W_PERM           TC_INVOKE_L
TC_QRY_WO_PERM          TC_RESULT_NL
TC_CONV_W_PERM          TC_RESULT_L
TC_CONV_WO_PERM         TC_U_ERROR
TC_RESPONSE             TC_L_CANCEL
TC_NO_RESPONSE          TC_U_CANCEL
TC_U_ABORT              TC_R_REJECT
TC_P_ABORT              TC_L_REJECT
TC_NOTICE               TC_U_REJECT
```

* `tc_user_id` (input)
  Specifies the TCAP user ID. This field is not used for sending TCAP components.

The following fields are variant-specific. Specify the appropriate field, depending on the variant of the SINAP/SS7 system you are using.

* `mtp_opc` (input)
  Specifies the originating point code of the node that initiated the message.

* `dialogue_id` (input/output)
  (CCITT/TTC/NTT/China) Specifies the dialogue ID for this dialogue. Use the dialogue ID returned from the call to `ca_get_dial_id()` or from within the `T_Block` returned by `ca_get_tc()`. If a single dialogue consists of multiple TCAP components, assign the same dialogue ID to each component.

* `trans_id` (input/output)
  (ANSI) Specifies the transaction ID for this transaction. Use the transaction ID returned from the call to `ca_get_trans_id()` or from within the `T_Block` returned by `ca_get_tc()`. If a single transaction consists of multiple TCAP components, assign the same transaction ID to each component.

The following fields are applicable to all network variants of the SINAP/SS7 system:

* `fwdlnk` (input)
  This field is used by TCAP to link the `T_Block` to an entry in the Dialogue or the Transaction ID table. You should not modify this field.

* `chp` (input/output)
  Specifies the `tc_chp_t` structure that contains component-handling information for the TCAP component. (For more information about the `tc_chp_t` structure, see "The Component-Handling Primitive Structure (`tc_chp_t`)" later in this section.)

The following fields are variant-specific. Specify the appropriate field, depending on the variant of the SINAP/SS7 system you are using.

* `dhp` (input/output)
  (CCITT/TTC/NTT/China) Specifies a `tc_dhp_t` structure that contains dialogue-handling information for the TCAP component. (For more information about the `tc_dhp_t` structure, see "The Dialogue-Handling Primitive Structure (`tc_dhp_t`)" later in this section.)

* `thp` (input/output)
  (ANSI) Specifies a `tc_thp_t` structure that contains transaction-handling information for the TCAP component. (For more information about this structure, see "The Transaction-Handling Primitive Structure (`tc_thp_t`)" later in this section.)

The following fields are applicable to all network variants of the SINAP/SS7 system:

* `buffer_type` (input)
  This field is set to `TBLK` at initialization. The field is internal to the SINAP/SS7 system and you should not modify it.

The Component-Handling Primitive Structure (`tc_chp_t`)

The `tc_chp_t` structure, which is used for all SINAP/SS7 variants, defines component-handling information. This structure is defined in the `tblock.h` include file and has the following format.

```
typedef struct tc_chp_s
{
    S16  linked_id;
    S16     corr_id;
    S16     invoke_id;
    U8      invoke_id_ind;
    U8      comp_type;
    U32  timer_value;
    U8      oper_class;
    U8      problem_type;
    U8      problem_code;
    U8      problem_specifier;
    BOOL    last_comp_ind;
    U16     extnd_data_size; /* contains data size in extended buffer */
    U8      dummy[1];
    U8      tot_data_len;
    U8      *extnd_data_ptr;/* When registered for XUDT/XUDTS */
                            /* points to extended data buffer */
#ifdef _LP_32_64_
    U32     filler;         /* For User32/Driver64 compatibility */
#endif  /* _LP_32_64_ */
    U8      data[MAX_DATA_SIZE_C]; /* data based on the primitive code*/
} tc_chp_t;
```

* `linked_id` (input/output)
  Links an operation invocation to a previous operation invocation.

* `corr_id` (input/output)
  Correlates an operation invocation to a previous operation invocation. The value of this field is in the range `0` through `255`. A value of `-1` indicates that the field is not applicable.

* `invoke_id` (input)
  Identifies an operation invocation. This field is used to indicate the MSU to which this TCAP component belongs. The value of this field is in the range `0` through `255`. A value of `-1` indicates that the field is not applicable.

* `invoke_id_ind` (output)
  Indicates whether `invoke_id` is valid. If this field is reset, the `invoke_id` is valid; otherwise, it is not. This field is used only in indication type primitives. A value of `INVALID_INVOKE_ID` indicates an invalid `invoke_id`.

* `comp_type` (output)
  Specifies the component type. A value of `HAND_OVER_COMP` indicates that the component is a handover component.

* `timer_value` (input/output)
  Specifies the maximum lifetime of the `invoke_id`. This field is applicable only when the value of `primitive_code` is `TC_INVOKE` (CCITT), `TC_INVOKE_L` (ANSI), or `TC_INVOKE_NL` (ANSI). The value of this timer should be less than the setting of the environment variable `TCAP_TQ_BINS` or the default value of half the sum of `MIN_TQ_BINS` and `MAX_TQ_BINS` which are defined in `tcglob.h`. For example: (4 + 3601)/2 = 1802.

* `oper_class` (input/output)
  Identifies the type of operating class. Valid values, which refer to constants defined in `tblock.h`, are as follows:

  1 – Report both success and failure
  2 – Report only failure
  3 – Report only success
  4 – Report neither success nor failure

  The TC user should specify `OPER_CLASS_1` (report both success and failure) while preparing a component for a TCAP message to be sent to TCAP, unless the application specifically requires the use of Class 2, 3, or 4.

* `problem_type` (input)
  Specifies a problem type tag. This field is valid only if the value of primitive_code is `TC_U_REJECT` or `TC_L_REJECT`.

  For the CCITT network variant, valid codes are:

Code	**Problem Type Indicated**
0x80	General
0x81	Invoke
0x82	Return result
0x83	Return error

  For the ANSI network variant, valid codes are:

Code	**Problem Type Indicated**
0x1	General
0x2	Invoke
0x3	Return result
0x4	Return error

* `problem_code` (input)
  Specifies a code associated with the problem type.

* `problem_specifier` (input)
  Specifies the value associated with the specified problem type. (See the `problem_type` parameter.) The appropriate values for the network variant being used and the problem type specified are listed in the following chart.

Problem Type	Value	Description
General Problem (GP)		
CCITT	0x00	Unrecognized component
	0x01	Mistyped component
	0x02	Badly structured component
ANSI	0x01	Unrecognized component
	0x02	Incorrect component portion
	0x03	Badly structured component portion
Invoke Problem (IP)		
CCITT	0x00	Duplicate invoke ID
	0x01	Unrecognized operation
	0x02	Mistyped parameter
	0x03	Resource limitation
	0x04	Initiating release
	0x05	Unrecognized linked ID
	0x06	Linked response unexpected
	0x07	Unexpected linked operation
ANSI	0x01	Duplicate invoke ID
	0x02	Unrecognized operation
	0x03	Incorrect parameter
	0x04	Unrecognized correlation ID
Return Result Problem (RRP)		
CCITT	0x00	Unrecognized invoke ID
	0x01	Returned result unexpected
	0x02	Mistyped parameter
ANSI	0x01	Unrecognized correlation ID
	0x02	Unexpected returned result
	0x03	Incorrect parameter

Problem Type	Value	Description
Return Error Problem (REP)		
CCITT		
	0x00	Unrecognized invoke ID
	0x01	Returned error unexpected
	0x02	Unrecognized error
	0x03	Unexpected error
	0x04	Mistyped parameter
ANSI		
	0x01	Unrecognized correlation ID
	0x02	Unexpected returned error
	0x03	Unrecognized error
	0x04	Unexpected error
	0x05	Incorrect parameter

* `last_comp_ind` (output)
  TCAP sets this field to indicate whether this is the last TCAP component of the MSU. This field is applicable only if the value of `primitive_code` is `TC_INVOKE` (CCITT), `TC_INVOKE_L` (ANSI), or `TC_INVOKE_NL` (ANSI).

* `*extnd_data_ptr` (input)
  This field points to the extended data buffer only if the TC user application is registered at the input boundary `SS7_INPUT_BOUNDARY_TCAPX`. For applications registered at the at the TCAPX boundary, CASL allocates 2048-byte component buffers (one per tblock). If your application is not registered at the TPACX boundary, set this field to 0.

* `extnd_data_size` (input)
  This field contains the data size in the extended buffer. This field is valid only if the application is registered at the input boundary `SS7_INPUT_BOUNDARY_TCAPX`. If your application is not registered at the input boundary, set this field to 0.

* `dummy` (output)
  This field is used as filler.

* `tot_data_len` (input/output)
  Specifies the total length of data in the `data` field.

* `data[MAX_DATA_SIZE]` (input/output)
  Provides the user data for the TCAP component, based on the value of `primitive_code`. As defined in the SINAP/SS7 `tblock.h` include file, this field provides up to 240 bytes of data for CCITT or 238 bytes of data for ANSI. For the CCITT/TTC/NTT/China variants of the SINAP/SS7 system, this field is formatted according to ITU-T (CCITT) Q.773 Recommendations. For the ANSI variant of the SINAP/SS7 system, this field is formatted according to ANSI T1.114.3 and T1.114.5 Recommendations. (`MAX_DATA_SIZE` is defined in the SINAP/SS7 `tblock.h` include file.)

Dialogue-Handling Primitive Structure (`tc_dhp_t`)

The `tc_dhp_t` structure, which is used for the CCITT/TTC/NTT/China variants of the SINAP/SS7 system, defines dialogue-handling information. This structure is defined in the `tblock.h` include file and has the following format. See the ITU-T (CCITT) Q.771 Recommendations for a description of the structure's fields.

```
typedef struc tc_dhp_s
{
        S16 msu_lost_count;               /* # of MSU lost by the driver */
        S16 write_free_mblk_count;        /* # of free M_Block, write queue */
        S16 read_free_mblk_count;         /* # of free M_Block, read queue */
        S16 read_queue_mblk_count;        /* # of M_Block in the read queue
                                          pending read */
/*****************************************************************************/
/* SCCP Called (destination address)/ Calling (origination address)*/
/* Party Address should be formatted as per the CCITT - Q713 */
/*****************************************************************************/

          U8      dest_addr[MAX_ADDR_LEN];/* destination TC-user */
          U8      orig_addr[MAX_ADDR_LEN];/* origination TC-user */
          U8      sccp_3rd_party_addr[MAX_ADDR_LEN];
          BOOL    comp_present_ind;       /* used in primitives of indication */
                                          /* type only */
                                          /* This field is set by the TCAP to */
                                          /* indicate that no component exist */
                                          /* for the dialogue */
          U32     alt_DPC;                /* alternate DPC for routing label */
          U8      tb_options;             /* bit-masked tblock options */
#define   USE_ALT_DPC             0x01    /* use alt_DPC in MTP routing label */
                                          /* else use DPC from dest_addr */
#define   USE_DEST_ORIG_ADDR      0x02    /* use dest-addr and orig_addr in */
                                          /* SCCP header of the TCAP message */
#define   use_alt_DPC      tb_options     /* backward compatibility */
          U8      qlty_of_svc;            /* acceptable quality of service */
          U8      dialogue_end_type;

/* if dialogue is aborted by the transaction Sub Layer (Local or Remote),
then following field contain P Abort Cause or for TC_NOTICE, it contains
information indicating the reason for the exception report, for example that
the message was returned by the SCCP with the reason as specified in Q.711. */

          U8 pa_report_cause;/* P Abort Cause or Report Cause */
/* if dialogue is aborted by TC-user, then following field contains
cause of abort and diagnostic information */
          U8      tot_ua_info_len;        /* user abort info length */
#define   MAX_UA_INFO_LEN_C       200
          U8 ua_info[MAX_UA_INFO_LEN_C];  /* user abort information */
          tc_association_tahp;            /* association handling part */
          U8      priority;               /* 0-3 priority values */
          U8      hop_count;              /* Hop count value to be inserted */
                                          /* into Extended Unit data */
                                          /* 0 = use default hop counter */
                                          /* >0 = insert this value in the
                                           hop counter parameter */
          U8       seq_control;           /* 0-31 slc values */
          U8       importance_parm;       /* ss7-2392: 1996 ITU-T Q.713 3.19 */
                                          /* The MSB is a flag to indicate if */
                                          /* SCCP Importance parm exists, if */
                                          /* it does, the 3 LSBs represents */
                                          /* the Importance values 0 to 7. */
} tc_dhp_t;
```

* `msu_lost_count` (input)
  Indicates the number of MSUs the driver has lost. This field is not used for sending TCAP components. The user application should initially set this to zero.

* `write_free_mblk_count` (input)
  Indicates the number of free `M_Blocks` in the write queue. This field is not used for sending TCAP components.

* `read_free_mblk_count` (input)
  Indicates the number of free `M_Blocks` in the read queue. This field is not used for sending TCAP components.

* `read_queue_mblk_count` (input)
  Indicates the number of `M_Blocks` in the read queue that are awaiting a read operation. This field is not used for sending TCAP components.

* `dest_addr[MAX_ADDR_LEN]` (input)
  Indicates the SCCP called-party address (the address of the destination TCAP user), which is formatted according to ITU-T (CCITT) Q.713 Recommendations. This field is applicable if the value of `primitive_code` is `TC_UNI` or `TC_BEGIN`. (`MAX_ADDR_LEN` is defined in the `tblock.h` include file.)

* `orig_addr[MAX_ADDR_LEN]` (input)
  Indicates the SCCP calling-party address (the address of the originating TCAP user), which is formatted according to ITU-T (CCITT) Q.713 Recommendations. This field is applicable if the value of `primitive_code` is `TC_UNI` or `TC_BEGIN`. (`MAX_ADDR_LEN` is defined in the `tblock.h` include file.)

* `sccp_3rd_party_addr[MAX_ADDR_LEN]` (input)
  For a TCP/IP agent (registered at the SCCP boundary) receiving messages from a TCAP application, this field specifies the original calling party address information. The TCP/IP agent overwrites the original SCCP called party address information with its own point code and pseudo SSN to establish a two-way dialogue with an application registered at the TCAP boundary on the same SINAP node and system. In this case, the TCP/IP agent requires the original SCCP calling party address to correctly format and route messages back to the originating node over TCP/IP.

  For a TCAP application (accessed through the TCP/IP agent) originating a dialogue (for CCITT variants) or transaction (for ANSI variants), the field specifies the SCCP called party address of the TCAP application. In this case, the called party address is required because the original called party address provided in the tblock and mblock is configured to address the own signaling point (OSP) code and pseudo SSN of the TCP/IP agent running on the same SINAP node.

  The CASL transparently copies the `sccp_3rd_party_addr` field between the `tblock` and `mblock` in both directions when sending and receiving `tblocks`. The

SINAP driver initializes this field in the `mblock` to zeros when the SINAP node receives messages from Level 2 of the SS7 network.

The constant specified in the `MAX_ADDR_LEN` parameter is defined in the include files `$SINAP_HOME/Include/mblock.h.` and `$SINAP_HOME/Include/tblock.h.`

* `comp_present_ind` (input)
  TCAP sets this field to indicate whether a component exists for the dialogue. This field is used only for `INDICATION` type primitives.

* `alt_DPC` (input)
  Indicates the alternate DPC that is used in the MTP routing label in place of the DPC of the destination address.

* `tb_options` (input)
  Indicates the bit-masked `tblock` options. See `tblock.h` for values of this field, such as `USE_ALT_DPC`.

* `qlty_of_svc` (input)
  Indicates the protocol class of service for handling this primitive and the return-on-error indicator. This field is applicable only if the value of `primitive_code` is `TC_UNI` or `TC_BEGIN`. Valid values are as follows:

Value	Description
`CONN_LESS_SVC_CLASS_0(0)`	Connectionless Class 0, no return on error
`CONN_LESS_SVC_CLASS_1(1)`	Connectionless Class 1, no turn on error
`0x80`	Connectionless Class 0, return on error
`0x81`	Connectionless Class 1, return on error

* `dialogue_end_type` (input)
  Indicates the way the dialogue is to end: `PREARRANGED_END (1)` indicates a prearranged end and `BASIC_END (2)` indicates a basic end. This field is applicable only if the value of `primitive_code` is `TC_END`.

* `pa_report_cause` (input)
  Indicates the `P_ABORT` or `REPORT` cause. See `tblock.h` for values of this field, such as `TSL_PA_TIMEOUT`.

* `tot_ua_info_len` (input)
  Indicates the length of the `ua_info` field. This field is applicable only if the value of `primitive_code` is `TC_U_ABORT`.

* `ua_info[MAX_UA_INFO_LEN_C]` (input)
  Indicates the reason for the abort; this field is applicable only if the value of `primitive_code` is `TC_U_ABORT`. If the dialogue was aborted by the TCAP user, this field contains the cause of the abort along with diagnostic information. (`MAX_UA_INFO_LEN` is defined in the `tblock.h` include file.)

* `ahp` (input)
  Specifies the `tc_association_t` structure that contains the application-context information for the MSU.

* `priority` (input)
  Indicates the message priority for the MSU. This parameter is valid only for SCCP Class 0 and Class 1 messages. The priority value can be in the range of 0 through 3 (lowest to highest).

* `hop_count` (input)
  Specifies the hop count value to be inserted into the MSU (XUDT only). The mandatory hop counter limits the number of global title translations (GTTs) that can be performed on the message. If the `hop_count` value is less than 1 or greater than 15, the SINAP/SS7 software defaults to the value 10. Any `hop_count` value between 1 and 15 is inserted into the MSU.

* `seq_control` (input)
  Indicates the value to use for the signaling link selection (SLS) field of the MSU's MTP routing label. This parameter is valid for SCCP Protocol Class 1 messages only. The valid value range for the TTC and NTT variants is 0 through 15. For all other variants excluding ANSI, the valid value range is 0 through 31.

  For the ANSI network variant, `seq_control` can have a value in the range of `0` through `255` if the SINAP user specified an eight-bit SLS through the `CHANGE-SLSTYPE` MML command. In all other cases the `seq_control` field can have a value in the range `0-31` (a five-bit SLS). See "SINAP/SS7 Interaction with the SS7 Network" in Chapter 2 for more information.

* `importance_parm` (input/output)
  For the CCITT (ITU-T) variant, if the environment variable SCCP_ITU96_IMPORTANCE_PARM is set, and the user registers at the TCAPX boundary, this field holds the importance parameter (ss7-2392: 1996 ITU-T Q.713 3.19). The MSB is used as a bit flag to indicate if the SCCP optional Importance parameter is included in the SCCP XUDT/XUDTS message. If it is, then the 3 LSBs represent Importance values 0 to 7.

The `tc_association_t` Structure

The `tc_association_t` structure contains the dialogue portion of the MSU. The dialogue portion defines the application-context name and optional user information to be used for the

application-context dialogue. The format of the `tc_association_t` structure is defined in the include file `$SINAP_HOME/Include/tblock.h` and has the following format.

```
typedef struct assoc
{
            int                 dlgInfoPresent;
            int                 pduType;
              acn_t               *applicationContextName;
#ifdef _LP_32_64_
            U32 filler; /* For User32/Driver64 compatibility*/
#endif  /* _LP_32_64_ */
            tc_user_data_t      userInformation;
            int                 abortSource;
            int                 result;
            int                 resultSourceDiag;
              int                 resultSourceDiagValue;
}tc_association_t;
```

* `dlgInfoPresent` (input)
  The TC user initializes this field to one of the following values to indicate whether the TCAP component contains a dialogue portion:
  - `TRUE` (`1`) indicates the presence of a dialogue portion.
  - `FALSE` (`0`) indicates that no dialogue portion is present.
* `pduType` (output)
  TCAP initializes this field to indicate the type of ACSE APDU present in the dialogue portion of the TCAP component. Possible values are `AARQ`, `AUDT`, `ABRT`, and `AARE`; see `tcap.h`.
* `applicationContextName` (input)
  The TC user initializes this field to the name of an `acn_t` structure that contains the application-context name for the dialogue. For more information, see "The acn_t Structure" later in this chapter.
* `userInformation` (input)
  The TC user initializes this field to the name of a `tc_user_data_t` structure that contains optional user information for the dialogue. For more information, see "The tc_user_data_t Structure" later in this chapter.
* `abortSource` (output)
  When an error occurs, TCAP initializes this field to one of the following values to indicate who aborted the dialogue:
  - `dialogue-service-provider` (`0`) indicates that the service provider aborted the dialogue, possibly due to a syntax error.

- `dialogue-service-user` (`1`) indicates that the TC user aborted the dialogue, possibly because the specified application-context name is not supported.

* `result` (output)
  TCAP initializes this field to one of the following values to indicate the status of the association request:

  - `accepted` (`0`) indicates that the association request has been accepted.
  - `reject-permanent` (`1`) indicates that the association request has been denied.

* `resultSourceDiag` (output)
  When an error occurs, TCAP initializes this field to a particular type of source diagnostic. Possible values are `TC_SERVICE_USER` and `TC_SERVICE_PROVIDER`.

* `resultSourceDiagValue` (input)
  When aborting a request, the service provider initializes this field to one of the following values:

  - `NULL` (`0`)
  - `no-reason-given` (`1`) can indicate a syntax error
  - `no-common-dialogue-portion` (`2`)

  When aborting a request, the TC user initializes this field to one of the following values:

  - `NULL` (`0`)
  - `no-reason-given` (`1`)
  - `application-context-name-not-supported` (`2`)

The acn_t Structure

The acn_t structure is the object identifier structure for the application context name of the application-context dialogue. The acn_t structure is defined in the include file $SINAP_HOME/Include/tblock.h and has the following format.

```
typedef struct
{
               long length;
               union {
                     char   *long_buf;
                     char   short_buf[MAX_ACN_SIZE];
               } b;
} acn_t;
```

When you design an application to initiate an application-context dialogue or respond to a TC-BEGIN message that contains a dialogue portion, make sure the application initializes the acn_t structure to the appropriate application-context name by using either of the following methods:

- The application can call the function tc_objmk(), a utility supplied with the CASL library, to create the object identifier (OID), then cast the results to an acn_t structure pointer, as shown in the following example. (Note that the acn_t structure stores the object identifier.)

```
ptblk->tc_user.dhp.ahp.applicationContextName =
     (acn_t *) tc_objmk(0x02, 0x04, 0x01, 0x01, 0x08,
     0x03, END_OF_OID);
```

- The application can initialize the fields of the acn_t structure to the application-context name. You **must** define the application-context name as a properly formatted ASE OID, encoded according to the rules documented in ITU-T Recommendation X.690, *Basic Encoding Rules*.

The tc_user_data_t Structure

The tc_user_data_t structure contains optional user information for the application-context dialogue (such as a password, application-initialization data, protocol-version information, and so on). This user information is exchanged independently of the remote-service operation and is transparent to TCAP. The tc_user_data_t structure is

defined in the `$SINAP_HOME/Include/tblock.h` include file and has the following format.

```
typedef struct tc_user_data_s
{
          int       size;
#if defined(__LP64__) || defined(_LP_32_64_)
          U32       filler; /* For User32/Driver64 compatibility */
#endif  /* __LP64__ || _LP_32_64_ */
          char      userInfo[MAX_USER_INFO];
}tc_user_data_t;
```

The fields in the `tc_user_data_t` structure are as follows:

* `size`
  The TC user initializes this field to the length of the `userInfo` field.

* `userInfo[MAX_USER_INFO]`
  The TC user initializes this field to define optional user information for the application-context dialogue. Format this field according to Section 4.2.3 of the 1993 edition of ITU-T Recommendation Q.773. According to Table 49 of the recommendation, this field must be defined as an ASN.1-encoded sequence of externals. (`MAX_USER_INFO` is defined in the `$SINAP_HOME/Include/tblock.h` include file.)

  You can use the ASN.1 compiler to properly format the value of this field. The compiler creates an ASN.1-encoded sequence of externals from the user information that you provide. You can then write the compilation results to the `userInfo` field.

The Transaction-Handling Primitive Structure (`tc_thp_t`)

The `tc_thp_t` structure, which is used for the ANSI variant of the SINAP/SS7 system, defines transaction-handling information. This structure is defined in the `tblock.h` include file and has the following format. See the ANSI T1.114.1 Recommendations for generic descriptions of this structure's fields.

```
typedef struct tc_thp_s
{
          S16      msu_lost_count;
          S16      write_free_mblk_count;
          S16      read_free_mblk_count;
          S16      read_queue_mblk_count;
          U8       dest_addr[MAX_ADDR_LEN];
          U8       orig_addr[MAX_ADDR_LEN];
          U8       sccp_3rd_party_addr[MAX_ADDR_LEN];
          U8       dest_tid[MAX_TID_SIZE];
          U8       orig_tid[MAX_TID_SIZE];
          U8       orig_tid_len;
          U8       local_tid[MAX_TID_SIZE];
          U8       local_tid_len;
          U8       packet_type;
          U8       qlty_of_svc;
          U8       seq_control;
          U8       hop_count;
          U8       priority;
          BOOL     comp_present_ind;
          U8       tb_options;
          U8       alt_DPC[DPC_LEN];
          U8       fictitious_OPC;
          U8       trans_end_type;
          U8       pa_report_cause;
          U8       tot_ua_info_len;
          U8       ua_info[MAX_UA_INFO_LEN];
} tc_thp_t;
```

* `msu_lost_count` (input/output)
  Indicates the number of MSUs the SINAP driver has lost. The user application should set this field to zero initially.

* `write_free_mblk_count` (output)
  Indicates the number of free `M_Blocks` in the SINAP driver write queue.

* `read_free_mblk_count` (output)
  Indicates the number of free `M_Blocks` in the read queue.

* `read_queue_mblk_count` (output)
  Indicates the number of `M_Blocks` in the read queue that are awaiting a SINAP driver read operation.

* `dest_addr[MAX_ADDR_LEN]` (input/output)
  Indicates the SCCP called-party address (the address of the destination TCAP user), which is formatted according to ANSI T1.113.3 Recommendations. This field is applicable only if the value of `primitive_code` is `TC_UNI`, `TC_QRY_W_PERM`, or `TC_QRY_WO_PERM`. (`MAX_ADDR_LEN` is defined in the SINAP/SS7 `tblock.h` include file.)

* `orig_addr[MAX_ADDR_LEN]` (input/output)
  Indicates the SCCP calling-party address (the address of the origination TCAP user), which is formatted according to ANSI T1.113.3 Recommendations. This field is applicable only if the value of the `primitive_code` parameter is `TC_UNI`, `TC_QRY_W_PERM`, or `TC_QRY_WO_PERM`. (`MAX_ADDR_LEN` is defined in the SINAP/SS7 `tblock.h` include file.)

* `sccp_3rd_party_addr[MAX_ADDR_LEN]` (input/output)
  For a TCP/IP agent (registered at the SCCP boundary) receiving messages from a TCAP application, this field specifies the original calling party address information. The TCP/IP agent overwrites the original SCCP called party address information with its own point code and pseudo SSN to establish a two-way dialogue with an application registered at the TCAP boundary on the same SINAP node and system. In this case, the TCP/IP agent requires the original SCCP calling party address to correctly format and route messages back to the originating node over TCP/IP.

  For a TCAP application (accessed through the TCP/IP agent) originating a dialogue (for CCITT variants) or transaction (for ANSI variants), the field specifies the SCCP called party address of the TCAP application. In this case, the called party address is required because the original called party address provided in the tblock and mblock is configured to address the own signaling point (OSP) code and pseudo SSN of the TCP/IP agent running on the same SINAP node.

  The CASL transparently copies the `sccp_3rd_party_addr` field between the `tblock` and `mblock` in both directions when sending and receiving `tblocks`. The SINAP driver initializes this field in the `mblock` to zeros when the SINAP node receives messages from Level 2 of the SS7 network.

  The constant specified in the `MAX_ADDR_LEN` parameter is defined in the include files `$SINAP_HOME/Include/mblock.h.` and `$SINAP_HOME/Include/tblock.h.`

* `dest_tid[MAX_TID_SIZE]` (input/output)
  Indicates the destination transaction ID. For a particular transaction ID, the TC user should save the destination address, the origination address, and this value. (`MAX_TID_SIZE` is defined in the SINAP/SS7 `tblock.h` include file.)

* `orig_tid[MAX_TID_SIZE]` (input/output)
  Indicates the origination transaction ID. (`MAX_TID_SIZE` is defined in the SINAP/SS7 `tblock.h` include file.)

* `orig_tid_len` (input/output)
  Indicates the length of the origination transaction ID. Valid values are `0` or `4`.

* `local_tid[MAX_TID_SIZE]` (output)
  Indicates the local transaction ID. (`MAX_TID_SIZE` is defined in the SINAP/SS7 `tblock.h` include file.)

* `local_tid_len` (output)
  Indicates the length of the local transaction ID. Valid values are `0` or `4`.

* `packet_type` (output)
  Indicates the TCAP packet type. A value of zero indicates that TCAP will translate the TC primitive to a TCAP packet type. A non-zero value indicates that the SINAP/SS7 system will use this field directly as a TCAP packet. See `tcap.h` for a list of nonzero packet types, such as `TSL_PT_UNI`.

* `qlty_of_svc` (input/output)
  Indicates the protocol class of service to use when sending this MSU and the return-on-error indicator. This field is applicable only if the value of `primitive_code` is `TC_UNI` or `TC_BEGIN`. Valid values are as follows:

Value	Description
`CONN_LESS_SVC_CLASS_0(0)`	Connectionless Class 0, no return on error
`CONN_LESS_SVC_CLASS_1(1)`	Connectionless Class 1, no return on error
`0x80`	Connectionless Class 0, return on error
`0x81`	Connectionless Class 1, return on error

* `seq_control` (input/output)
  Indicates the signaling link selection (SLS) code of the link over which messages are sent.

  For quality of service requiring Class 0 (`CONN_LESS_SVC_CLASS_0`), set the `seq_control` field to 0. For quality of service requiring Class 1 (`CONN_LESS_SVC_CLASS_1`), set the `seq_control` field to the appropriate service link selection (SLS) value (0 through 31).

* `hop_count` (output)
  Specifies the hop count value to be inserted into the MSU. (XUDT only) The mandatory hop counter limits the number of global title translations (GTTs) that can be performed on the message. The valid range of values for the hop count is 1 through 15. The default count is 12.

* `priority` (input/output)
  Indicates the transaction priority (in the range `0` to `3`) for each transaction-handling primitive. The priority of the MSU's SIO octet is based on this value.

* `tb_options` (input)
  Indicates one of the following bit-masked `tblock` option codes to use for message routing.

Code	Description
0x01	Use the `alt_DPC` in the MTP routing label if set. Otherwise, use the DPC specified in the `dest_addr` field.
0x02	Use the destination address specified in the `dest_addr` field and the origination address specified in the `orig_addr` field of the SCCP header of the TCAP message. (Used for backward compatibility.)

* `comp_present_ind` (output)
  Indicates whether the transaction contains a component: `0` indicates that there is no component present; `1` indicates that there is a component present. This field is used for `INDICATION` type primitives only.

* `alt_DPC[DPC_LEN]` (output)
  Indicates the alternate DPC that is used in the MTP routing label in place of the DPC of the destination address. (`DPC_LEN` is defined in the SINAP/SS7 `tblock.h` include file.)

* `fictitious_OPC` (input/output)
  Indicates whether the MTP routing label contains the point code of the originating address or a fictitious OPC. A value of `1` indicates the MTP routing label contains a fictitious OPC. Otherwise, the MTP routing label uses the OPC in `orig_addr`.

* `trans_end_type` (input/output)
  Indicates how the transaction is to be ended: `PREARRANGED_END(1)` indicates a prearranged end and `BASIC_END(2)` indicates a basic end. This field is applicable if the value of `primitive_code` is `TC_QRY_W_PERM`.

* `pa_report_cause` (output)
  Indicates the `P_ABORT` or `REPORT` cause. If the user aborts the transaction, this field contains the cause of the abort, along with diagnostic information. See `tcap.h` for possible values of this field, such as `TSL_PA_TIMEOUT`.

* `tot_ua_info_len` (output)
  Indicates the length of the `ua_info` field. This field is applicable only if the value of `primitive_code` is `TC_U_ABORT`.

* `ua_info [MAX_UA_INFO_LEN_A]` (output)
  Indicates the reason for the abort. If the user aborts the transaction, this field contains the cause of the abort along with diagnostic information. This field is applicable only if the value of `primitive_code` is `TC_U_ABORT`.

FILES

`$SINAP_HOME/Include/arch.h`, `ca_error.h`

RETURN VALUES

The `ca_get_tc()` function returns an index to the next available `T_Block`. If the function returns `-1`, there is an error; see `errno` for error number and description. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

A possible CASL value for `errno` follows.

Value	Meaning
`CA_ERR_NO_MSUS`	There are no MSUs in the batch buffer.

The TCAP can return the following errors.

Error	Action
`TC_ERR_NOT_REG_AT_TCAP_BOUNDARY`	Reregister the application at the TCAP boundary using `ss7_input_boundary= SS7_INPUT_BOUNDARY_TCAP`. The SINAP/SS7 system provides an immediate return; `ca_get_tc()` is not executed.
`TC_ERR_OUT_OF_TRANS_ID`	Increase the value of the `max_trans_id` parameter of the `ca_register()` function to reregister the application with a greater number of transaction IDs. The SINAP/SS7 system deallocates the `T_Block`; `ca_get_tc()` is not executed. `TC_CONTINUE` (CCITT) or `TC_CONV_W_PERM` and `TC_CONV_WO_PERM` (ANSI); `TC_END` (CCITT) or `TC_RESPONSE` (ANSI); and `TC_P_ABORT` and `TC_U_ABORT` primitives do not return this error.

Error	Action
TC_ERR_T_BLOCK_CAPACITY_EXHAUSTED	Increase the value of the `tc_count` parameter of the `ca_register()` function to reregister the application with a greater number of `T_BLOCK`s. The SINAP/SS7 system deallocates the `T_Block`; `ca_get_tc()` is not executed.
TC_ERR_INV_TSL_STATE	Report this error, along with all debug information, to the Customer Assistance Center (CAC). The SINAP/SS7 system provides an immediate return; `ca_get_tc()` is not executed.
TC_ERR_INV_TSL_EVENT	Report this error, along with all debug information, to the CAC. SINAP/SS7 provides an immediate return; `ca_get_tc()` is not executed.
TC_ERR_INV_ISM_STATE	Report this error, along with all debug information, to the CAC. SINAP/SS7 provides an immediate return; `ca_get_tc()` is not executed.
TC_ERR_INV_ISM_EVENT	Report this error, along with all debug information, to the CAC. SINAP/SS7 provides an immediate return; `ca_get_tc()` is not executed.
TC_ERR_TSL_TEQ_OVERFLOW	Report this error, along with all debug information, to the CAC. SINAP/SS7 provides an immediate return; `ca_get_tc()` is not executed.
TC_ERR_ISM_TEQ_OVERFLOW	Report this error, along with all debug information, to the CAC. SINAP/SS7 provides an immediate return; `ca_get_tc()` is not executed.
TC_ERR_PTR_TO_USF_NOT_SET	No user-defined pointer.
TC_ERR_OUT_OF_TC_USER_ID	A user-supplied function has exhausted its supply of user IDs. Check to ensure that the supply of IDs is large enough.
TC_ERR_TRANS_ID_NOT_ASSIGNED	Before calling `ca_get_tc()`, the process must call the `ca_get_trans_id()` function to obtain a transaction ID for the transaction.

Error	Action
TC_ERR_TCAP_OWN_TRANS_ID	The transaction is not under application control. A TC_RESPONSE primitive will release this transaction. (As per ANSI Recommendations, a TC_NO_RESPONSE primitive, which is prearranged by both TC users, causes messages to be discarded rather than being sent to the network.) No further action is required.

The `ca_get_tc()` function calls the `ca_alloc_tc()` function and can also return the errors listed under that function. Under certain circumstances `ca_get_tc()` may call `ca_dealloc_tc()`; therefore, it may return the errors listed under `ca_dealloc_tc()`.

SEE ALSO

`ca_alloc_tc()`, `ca_dealloc_tc()`, `ca_process_tc()`, `ca_put_tc()`

ca_rel_dial_id()

SYNOPSIS

```
S32 ca_rel_dial_id(
          S32 dial_id);
```

DESCRIPTION

The `ca_rel_dial_id()` function lets a user release an allocated dialogue ID after the dialogue session is over. This function is for CCITT, TTC, NTT, and China applications; for ANSI applications, use the `ca_rel_trans_id()` function instead.

PARAMETERS

* `dial_id` (input)
  Specifies the dialogue ID to be released.

FILES

`arch.h`, `ca_error.h`

DIAGNOSTICS AND WARNINGS

The `ca_rel_dial_id()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning. See `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and meaning.

The TCAP can return the following errors.

Error	Action
TC_ERR_NOT_REG_AT_TCAP_BOUNDARY	Reregister the application at the TCAP boundary using ss7_input_boundary= SS7_INPUT_BOUNDARY_TCAP.
TC_ERR_DIAL_ID_ALREADY_RELEASED	This is an informational message. No action is required.
TC_ERR_TCAP_OWN_DIAL_ID	The dialogue is not under application control. An END message will release this dialogue. (As per Q.775 a prearranged END will not cause messages to be sent to the network.) No further action required.
TC_ERR_INV_DIAL_ID	The dialogue ID is less than 0 or greater than the max_dialogue_id minus 1. The application should use an ID in the correct range.
TC_ERR_DIAL_ID_NOT_ASSIGNED	This is an informational message. No action is required.

SEE ALSO

ca_alloc_tc(), ca_dealloc_tc(), ca_get_dial_id(), ca_get_tc(), ca_process_tc(), ca_put_tc()

ca_rel_trans_id()

SYNOPSIS

```
int     ca_rel_trans_id(
          S32 trans_id);
```

DESCRIPTION

The `ca_rel_trans_id()` function releases a transaction ID and returns it to the pool of available transaction IDs, from which it can be assigned to another transaction. This function is for ANSI applications; for CCITT, TTC, NTT, and China applications, use the `ca_rel_dial_id()` function instead.

To effectively manage the supply of transaction IDs, the calling process should call `ca_rel_trans_id()` after terminating the transaction to which the transaction ID was assigned.

PARAMETERS

* `trans_id` (input)
  Specifies the transaction ID to be released.

FILES

`arch.h`, `ca_error.h`, `tblock.h`

RETURN VALUES

The `ca_rel_trans_id()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

The TCAP can return the following error.

Error	Action
TC_ERR_TRANS_ID_ALREADY_RELEASED	Use a TC_QRY_W_PERM or TC_QRY_W_PERM primitive to initiate a transaction in TCAP. The SINAP/SS7 system provides an immediate return; ca_rel_trans_id() is not executed.

SEE ALSO

ca_get_tc(), ca_get_trans_id(), ca_put_tc()

IPC Functions

This section contains an alphabetic reference of the following CASL functions, which an application uses to perform interprocess communications (IPC) with other applications or SINAP/SS7 processes.

- `ca_ascii_u32()`
- `ca_cancel_def()`
- `ca_check_key()`
- `ca_get_key()`
- `ca_get_msg()`
- `ca_put_cmd()`
- `ca_put_msg()`
- `ca_put_msg_def()`
- `ca_put_reply()`
- `ca_restart_timer()`
- `ca_swap_keys()`
- `ca_u32_ascii()`

ca_ascii_u32()

SYNOPSIS

```
int     ca_ascii_u32(
          char    *pnode,
          char    *pmod,
          char    *papp,
          char    *pproc,
          ipc_key_t *pipc_key);
```

DESCRIPTION

The `ca_ascii_u32()` function converts the ASCII representations of node, module, application, and process names to 32-bit, unsigned integers. The function then assigns these integers to the `node`, `module`, `appl`, and `proc` fields of the `ipc_key_t` structure. (The `ipc_key_t` structure, also called the *IPC key*, is defined in the `sinap.h` include file.)

PARAMETERS

* `pnode` (input)
  Specifies a pointer to the node name of the calling process. The name can be up to four bytes long in an ASCII string. Currently, the SINAP/SS7 system does not use this parameter.

* `pmod` (input)
  Specifies a pointer to the module name of the calling process. The name can be up to four bytes long in an ASCII string. Currently, the SINAP/SS7 system does not use this parameter.

* `papp` (input)
  Specifies a pointer to the application name of the calling process. This name can be up to four bytes long in an ASCII string.

* `pproc` (input)
  Specifies a pointer to the process name of the calling process. This name can be up to four bytes long in an ASCII string.

* `pipc_key` (output)
  Specifies a pointer to the process's IPC key. (The IPC key is defined by the `ipc_key_t` structure, which is described in the following section.)

IPC Key Structure (`ipc_key_t`)

The `ipc_key_t` structure contains the following fields and is defined in the include file `sinap.h`. (The `ca_ascii_u32()` function assigns values to the `node`, `module`, `appl`, and `proc` fields.)

```
typedef struct ipc_key_s
{
      U32       node;
      U32       module;
      U32       appl;
      U32       proc;
      U8        instance;
      U8        node_index;
      U16       ipc_index;
} ipc_key_t;
```

* `node` (output)
  Specifies the ID of the SINAP node on which your application is running. You can determine this value from the `NODE=` entry in the `/etc/sinap_master` file. You should modify any script files or user-defined program files that contain an invalid, hard-coded node name.

* `module` (output)
  Specifies the name or ID of the module (system). You can determine this value from the `MODULE=` entry in the `/etc/sinap_master` file. You should modify any script files or user-defined program files that contain an invalid, hard-coded module name. For this version of the SINAP/SS7 system, the default name of the module is `M1` and its value is `0`.

* `appl` (output)
  Specifies the compressed application ID.

* `proc` (output)
  Specifies the compressed process ID.

* `instance` (output)
  Specifies the instance ID (in the range `1` to `8`). A value of `0` indicates that the field is not used.

* `node_index` (output)
  Specifies the index (0 through 3) of the node.

* `ipc_index` (output)
  Specifies the index ID of the IPC process table.

FILES

`arch.h`, `ca_error.h`, `sinap.h`

RETURN VALUES

The `ca_ascii_u32()` function can return the following value.

Value	Meaning
`0`	Successful.

SEE ALSO

`ca_check_key()`, `ca_get_key()`, `ca_swap_keys()`, `ca_u32_ascii()`

ca_cancel_def()

SYNOPSIS

```
int     ca_cancel_def(
          U32     timer_id);
```

DESCRIPTION

The `ca_cancel_def()` function cancels the timer specified by the parameter `timer_id`. All pending deferred messages that have this timer ID are also discarded.

After a call to this function, timeout messages might still appear because of a race condition.

PARAMETERS

* `timer_id` (input)
  Specifies the 32-bit ID of the timer to be cancelled. This is the same value that you assigned to the `timer_id` parameter of the `ca_put_msg_def()` function.

FILES

`arch.h`, `ca_error.h`

RETURN VALUES

The `ca_cancel_def()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

SEE ALSO

`ca_get_msg()`, `ca_put_msg()`, `ca_put_msg_def()`, `ca_restart_timer()`

ca_check_key()

SYNOPSIS

```
int     ca_check_key(
          ipc_key_t      destn_key);
```

DESCRIPTION

The `ca_check_key()` function checks whether an IPC key is valid. If the IPC key is not valid, the function returns an error.

PARAMETERS

* `destn_key` (input)
  Specifies the `ipc_key_t` structure that contains the IPC key of the destination process. Before calling `ca_check_key()`, verify values have been assigned to the fields in the `ipc_key_t` structure, which is described in the following section.

IPC Key Structure (`ipc_key_t`)

The `ipc_key_t` structure contains the following fields and is defined in the include file `sinap.h`.

```
typedef struct ipc_key_s
{
          U32          node;
          U32          module;
          U32          appl;
          U32          proc;
          U8           instance;
          U8           node_index;
          U16          ipc_index;
} ipc_key_t;
```

* `node` (output)
  Specifies the ID of the SINAP node on which your application is running. You can determine this value from the `NODE=` entry in the `/etc/sinap_master` file. Modify any script files or user-defined program files that contain an invalid, hard-coded node name.

* `module` (output)
  Specifies the name or ID of the module (system). You can determine this value from the `MODULE=` entry in the `/etc/sinap_master` file. Modify any script files or user-defined program files that contain an invalid, hard-coded module name.

* `appl` (input)
  Specifies the compressed application ID.

* `proc` (input)
  Specifies the compressed process ID.

* `instance` (input)
  Specifies the instance ID (`1` to `8`). A value of `0` indicates the field is unused.

* `node_index` (input)
  Specifies the index (0 through 3) of the node.

* `ipc_index` (input)
  Specifies the index ID of the IPC process table.

FILES

`arch.h`, `ca_error.h`, `sinap.h`

RETURN VALUES

The `ca_check_key()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

The CASL value for `errno` is as follows.

Value	Meaning
`CA_ERR_DESTN_KEY`	Destination process key not found or invalid.

SEE ALSO

`ca_ascii_u32()`, `ca_get_key()`, `ca_swap_keys()`, `ca_u32_ascii()`

ca_get_key()

SYNOPSIS

```
int      ca_get_key(
           char        *pnode,
           char        *pmodule,
           char        *pappl,
           char        *pproc,
           U32         inst,
           ipc_key_t   *pipc_key);
```

DESCRIPTION

The `ca_get_key()` function retrieves the IPC key of an process. At registration, each client application process declares itself as part of some application and as an instance of some process. This information is stored in an `ipc_key_t` structure (or *IPC key*). The SINAP/SS7 system uses the information in the IPC key to identify the process and to direct messages to it. Therefore, when an application process wants to initiate interprocess communications with another process, it must call this function to obtain the IPC key of that process.

NOTE

Interprocess communications do not use the SS7 network, as do communications initiated by the `ca_put_msu()` or `ca_put_tc()` function.

PARAMETERS

* `pnode` (input)
  Specifies a pointer to the node name of the process to be called. Valid values are `N1` and `0`. If you specify a value of `0`, the value of the environment variable `SINAP_NODE` is used.

* `pmodule` (input)
  Specifies a pointer to the module name of the process to be called. Valid values are `M1` and `0`. If you specify a value of `0`, the value of the environment variable `SINAP_MODULE` is used.

* `appl` (input)
  Specifies a pointer to the application name of the process to be called. This name can be up to four bytes long in an ASCII string.

* `pproc` (input)
  Specifies a pointer to the process name of the process to be called. This name can be up to four bytes long in an ASCII string.

* `inst` (input)
  Specifies a pointer to the logical instance ID of the process to be called. The value of this parameter can be from `0` (if not used) to `16`. If the value of `inst` is `0`, the IPC key of the first instance is returned.

* `pipc_key` (output)
  A pointer to the `ipc_key_t` structure for the specified application process. This structure contains the IPC key of that process. (The `ipc_key_t` structure is described in the following section.)

IPC Key Structure (`ipc_key_t`)

The `ipc_key_t` structure contains the following fields and is defined in the include file `sinap.h`.

```
typedef struct ipc_key_s
{
          U32      node;
          U32      module;
          U32      appl;
          U32      proc;
          U8       instance;
          U8       node_index;
          U16      ipc_index;
} ipc_key_t;
```

* `node` (output)
  Specifies the ID of the SINAP node on which your application is running. You can determine this value from the `NODE=` entry in the `/etc/sinap_master` file. You should modify any script files or user-defined program files that contain an invalid, hard-coded node name.

* `module` (output)
  Specifies the name or ID of the module. You can determine this value from the `MODULE=` entry in the `/etc/sinap_master` file. You should modify any script files or user-defined program files that contain an invalid, hard-coded module name. For this release of the SINAP/SS7 system, the default module name is M1.

* `appl` (output)
  Specifies the compressed application ID.

* `proc` (output)
  Specifies the compressed process ID.

* `instance` (output)
  Specifies the instance ID (in the range `1` to `8`). A value of `0` indicates that the field is not used.

* `node_index` (output)
  Specifies the index (0 through 3) of the node.

* `ipc_index` (output)
  Specifies the index ID of the IPC process table.

FILES

`arch.h`, `ca_error.h`, `sinap.h`

RETURN VALUES

The `ca_get_key()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

Possible UNIX values for `errno` are as follows.

Value	Meaning
`EBADF`	An invalid open file descriptor was specified.
`ENOTTY`	This `fides` is not associated with a device driver that accepts control functions.
`EFAULT`	The pointer to the specified message is outside the address space allocated to the process.
`EINVAL`	Queue ID is not a valid message queue ID. The value of `msg_type` is less than `1`, or `msg_sz` is greater than `0` or the system-imposed limit.

Value	Meaning
EIO	An I/O error occurred during a read or write operation.
ENXIO	The requested service cannot be performed on this particular subdevice.
ENOLINK	The link to a requested machine is no longer active.

Possible CASL values for `errno` are as follows.

Value	Meaning
CA_ERR_APPL	Application name invalid.
CA_ERR_DESTN_KEY	Destination process key not found.
CA_ERR_INST	Instance out of range.
CA_ERR_PROC	Process name invalid.

SEE ALSO

`ca_ascii_u32()`, `ca_check_key()`, `ca_swap_keys()`, `ca_u32_ascii()`

ca_get_msg()

SYNOPSIS

```
int  ca_get_msg(
       long       msg_type,
       i_block_t  *piblk,
       U16        max_sz,
       BOOL       fwait);
```

DESCRIPTION

The `ca_get_msg()` function receives a message from the client process's IPC queue. If there are no messages, the function returns an error.

PARAMETERS

* `msg_type` (input)
  Specifies the type of message being received from the IPC queue of the client process. If the value of this parameter is `0`, the IPC queue is treated as a first-in first-out (FIFO) queue. If the value of this parameter is greater than `0`, the oldest message on the queue of that type is returned.

* `piblk` (output)
  Specifies a pointer to the `I_Block`, if a message exists. For a description of the `I_Block`, see the following section, "The Main `I_Block` Structure (`i_block_t`)."

* `max_sz` (input)
  Specifies the maximum size of the data to be received from the IPC queue. If the incoming message is larger than this value, `ca_get_msg()` truncates the message and returns `0`. Thus, the calling process should check the length of the field in the `I_Block` header and the total size of the buffer to determine whether the incoming message is truncated.

* `fwait` (input)
  Specifies whether `ca_get_msg()` is to wait for a message. Specify a `1` to execute the call in blocking-mode (wait for a message); otherwise, specify `0` to execute the call in non-blocking mode (return if no message).

Main `I_Block` Structure (`i_block_t`)

The following fields are set in the `i_block_t` structure, which is defined in the include file `iblock.h`. The `iblock.h` include file defines the structure of messages (`I_Blocks`) sent

by means of IPC. An `I_Block` is composed of a CASL control part, a transaction part, a timestamp, a node ID, an originator key, a destination key, and a message body.

```
typedef  struct  i_block_s
{
    ca_ctrl_t       ca_ctrl;
    ipc_trans_t     trans;
    timestamp_t     ts;
    node_id_t       node;
    ipc_key_t       orig_id;
    ipc_key_t       dest_id;
    ipc_data_t      msg;
} i_block_t;
```

* `ca_ctrl` (output)
  Specifies the CASL control structure for this `I_Block`. For information about this structure, see "The CASL Control Structure (`ca_ctrl_t`)" later in this section.

* `trans` (output)
  Specifies the transaction ID structure. For information about this structure, see "The IPC Transaction ID Structure (`ipc_trans_t`)" later in this section.

* `ts` (output)
  Specifies a collection of timestamps that the SINAP/SS7 system automatically inserts. The timestamps aid monitoring and logging, and are visible when you run the BITE log-analysis program. (For a description of `timestamp_t` structure, see "The Timestamp Structure (`timestamp_t`)" later in this section.

* `node` (output)
  Specifies the ID of the SINAP node. This structure is internal to the SINAP/SS7 system and should not be modified.

* `orig_id` (output)
  Specifies the `ipc_key_t` structure that contains the IPC key for a sender application process. For information about the `ipc_key_t` structure, see "The IPC Key Structure (`ipc_key_t`)" later in this section.

* `dest_id` (output)
  Specifies the `ipc_key_t` structure that contains the IPC key for the intended destination process of the `I_Block`. You can obtain this IPC key by calling the `ca_get_key()` function. For information about the `ipc_key_t` structure, see "The IPC Key Structure (`ipc_key_t`)" later in this section.

* `msg` (output)
  Specifies the `ipc_data_t` structure that contains the IPC user data. For information about the `ipc_data_t` structure, see "The IPC Data Structure (`ipc_data_t`)" later in this section.

CASL Control Structure (`ca_ctrl_t`)

The following fields make up the `ca_ctrl_t` structure, which is defined in the include file `blkhdr.h`.

```
typedef struct   ca_ctrl_s
{
       int     msg_type;       /* For compatibility with the existing UNIX IPC
                                     mechanism. */
       int     msu_cnt;        /* # of MSUs pending */
       int     free_cnt;       /* # of free MSUs in read queue */
       int     wfree_cnt;      /* # of free MSUs in write queue */
       S16     lost_cnt;       /* # of MSUs lost due to insuff. resources */
       S16     data_size;      /* Total size of structure excluding this
                                  structure. */
       U8      node_index;     /* index (0 - 3) of current node */
       U8      sinap_variant;  /* V_CCITT, V_ANSI, V_HYBRID, V_TTC */
       U16     link;           /* index of the origination link */
       pid_t   pid;            /* Process ID of a specific process or 0
                                  for load distribution */
       int     msg_sender;     /* Set to 0 if from link otherwise contains
                                  the process ID */
       U8      iblk;           /* TRUE if data contains I_Block */
                               /* Not used anywhere!!!!! */
       U8      rw;             /* Flag for monitor message only */
       U8      monitor_id;     /* monitor ID for monitored MSU  */
       U8      ssn_sio;        /* SSN or SIO ID for load distribution. */
       struct {
               U32     timer_id;       /* For CASL internal use only */
               U32     timer_val;      /* For CASL internal use only */
               int     omsg_type;      /* For CASL internal use only */
       } timr;
       struct {                        /* For internal use only */
               int     source;         /* For distribution managment. */
               int     destination;    /* For protocol processing. */
#ifdef _KERNEL
               mblk_t  *mptr;          /* Back pointer to mblk_t.  L3 only.*/
#else
               void    *mptr;          /* ss7-1102 - dummy back pointer. */
#ifdef _LP_32_64_
               U32  filler;            /* For User32/Driver64 compatibility*/
#endif  /* _LP_32_64_ */
#endif  /* _KERNEL */
       } internal;
} ca_ctrl_t;
```

* `msg_type` (output)
  Specifies the type of MSU being sent. This field is compatible with the existing UNIX IPC mechanism.

* `data_size` (output)
  Specifies the total size of the structure, excluding this field.

* `node_index` (output)
  This is an internal field that is automatically initialized to the appropriate value for the SINAP node.

* `sinap_variant` (output)
  This is an internal field that is automatically initialized to the appropriate value for the network variant being used on the SINAP node.

* `lost_cnt` (output)
  Specifies the number of `M_Blocks` lost due to insufficient resources within the SS7 driver.

* `msu_cnt` (output)
  Specifies the number of MSUs pending.

* `free_cnt` (output)
  Specifies the number of free MSUs pending in the read queue.

* `wfree_cnt` (output)
  Specifies the number of free MSUs pending in the write queue.

The following fields are internal to the SINAP/SS7 system and you should not modify them:

* `pid` (output)
* `link` (output)
* `msg_sender` (output)
* `iblk` (output)
* `rw` (output)
* `monitor_id` (output)
* `ssn_sio` (output)
* `source` (output)
* `destination` (output)
* `mptr` (input) Specifies a pointer to `m_block_t`, Level 3.
* `timer_id` (output)
* `timer_val` (output)
* `omsg_type` (output)

IPC Transaction ID Structure (`ipc_trans_t`)

The `ipc_trans_t` structure contains the following fields and is defined in the include file `iblock.h`.

```
typedef  struct  ipc_trans_s
{
         int     msg_type;
         U32     ref_nbr;
         U16     rw_ind;
         U8      monitor_id;
         U8      scenario_id;
} ipc_trans_t;
```

* `msg_type` (output)
  Specifies the basic message function identifier that the SINAP/SS7 system and client applications use to identify a message. When defining client application messages, you should specify message types within the range of `CL_IPC_MIN` and `CL_IPC_MAX` (see the include file `iblock.h` for more information).

* `ref_nbr` (output)
  Specifies a reference number that allows the sending and receiving processes to keep track of messages that are of the same type. The reference number lets the receiving process associate a reply with an instance of a command.

* `rw_ind` (output)
  Specifies a read or write indicator for the monitor. This field is internal to the SINAP/SS7 system; you should not modify it.

* `monitor_id` (output)
  Associates the IPC message with a particular BITE monitor session. This field is internal to the SINAP/SS7 system; you should not modify it.

* `scenario_id` (output)
  Associates the IPC message with a particular BITE scenario execution session. This field is internal to the SINAP/SS7 system; you should not modify it.

Timestamp Structure (`timestamp_t`)

The `timestamp_t` structure contains the following fields and is defined in the include file `timestamp.h`.

```
typedef struct timestamp_s
{
         U16     index;
         stamp_t stamp[MAX_TIME_STAMPS];
} timestamp_t;
```

* `index` (output)
  Specifies the next slot to stamp the time.

* `stamp[MAX_TIME_STAMPS]` (output)
  Specifies the timestamp slots. For a description of the structure in which these timestamps are stored, see "The `stamp_t` Structure" below. (`MAX_TIME_STAMPS` is defined in the SINAP/SS7 `timestamp.h` include file.)

The `stamp_t` Structure

The `stamp_t` structure contains the following fields and is defined in the include file `timestamp.h`.

```
typedef struct stamp_s
{
       U32     secs;                   /* time in seconds since 1/1/70 */
       U8      tsid;                   /* timestamp id                 */
       U8      ipcx;                   /* ipc index if applicable      */
       U16     msec;                   /* time in milliseconds         */
} stamp_t;
```

* `secs` (output)
  Specifies the time (in seconds) since 1/1/70.

* `tsid` (output)
  Specifies the timestamp ID. These IDs are defined in the include file `timestamp.h`.

* `ipcx` (output)
  Specifies the IPC index, if applicable.

* `msec` (output)
  Specifies the time, in milliseconds.

Node ID Structure (node_id_t)

The node_id_t structure contains the following fields and is defined in the include file iblock.h.

```
 typedef struct node_id_s
{
         U8      ni;
         U32     spc;
} node_id_t;
```

* ni (output)
  Specifies the network indicator.

* spc (output)
  Specifies the signaling point code.

IPC Key Structure (ipc_key_t)

The following fields make up the ipc_key_t structure, which is defined in the include file sinap.h.

```
typedef struct ipc_key_s
{
          U32      node;
          U32      module;
          U32      appl;
          U32      proc;
          U8       instance;
          U8       node_index;
          U16      ipc_index;
} ipc_key_t;
```

* node (output)
  Specifies the ID of the SINAP node on which your application is running. You can determine this value from the NODE= entry in the /etc/sinap_master file. You should modify any script files or user-defined program files that contain an invalid, hard-coded node name.

* module (output)
  Specifies the name or ID of the module. You can determine this value from the MODULE= entry in the /etc/sinap_master file. You should modify any script files or user-defined program files that contain an invalid, hard-coded module name.

* `appl` (output)
  Specifies the compressed application ID.

* `proc` (output)
  Specifies the compressed process ID.

* `instance` (output)
  Specifies the instance ID (in the range `1` through `8`). A value of `0` indicates that the field is not used.

* `node_index` (output)
  Specifies the index (0 through 3) of the node.

* `ipc_index` (output)
  Specifies the index ID of the IPC process table.

IPC Data Structure (`ipc_data_t`)

The following fields make up the `ipc_data_t` structure, which is defined in the include file `iblock.h`.

```
typedef  struct   ipc_data_s
{
         U8      more_ind;
         U32     len;
         U32     ret_code;
#if defined(__LP64__) || defined(_LP_32_64_)
         U32     filler; /* For User32/Driver64 compatibility */
#endif /* __LP64__ || _LP_32_64_ */
} ipc_data_t;
```

* `more_ind` (output)
  Specifies whether an IPC message is the last in a sequence. This field is useful if the data portion of a message exceeds the maximum amount of data that UNIX can send in a single data packet. Note that the limit is an UNIX configuration parameter, initially set to 4096 octets. The SINAP/SS7 system also uses `more_ind` when a command reply exceeds the response timeout. In this case, a command reply could consist of an arbitrary number of messages and a final reply; the `more_ind` field would be set to `1` to indicate that the receiving process is working on the reply. The final reply would indicate the result of the command.

* `len` (output)
  Specifies the length (in octets) of the data portion of the message body.

* `ret_code` (output)
  Specifies a return code value. By returning a user-defined value, a client application can use this field to indicate success or failure.

NOTE

The data portion of a message should follow the message field of `i_block_t`. The structure of the data portion is dependent on the `msg_type` field. The following use of `i_block_t` is recommended.

```
typedef struct user_struc_s
{
  i_block_t   iblk_hdr;
  char        user_data[MAX_IBLK_DATA_SZ];
} user_struc_t;
```

FILES

`arch.h`, `ca_error.h`, `iblock.h`, `sinap.h`, `sysdefs.h`, `sys/time.h`, `timestamp.h`

RETURN VALUES

The `ca_get_msg()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

Possible UNIX values for `errno` are as follows.

Value	Meaning
`EBADF`	An invalid open file descriptor was specified.
`ENOTTY`	This `fides` is not associated with a device driver that accepts control functions.
`EINVAL`	Queue ID is not a valid message queue ID. The value of `msg_type` is less than `1`, or `msg_sz` is less than `0` or greater than the system-imposed limit.
`EACCES`	Operation permission is denied to the calling process.
`E2BIG`	The message text is greater than the maximum size allowed.

Value	Meaning
ENOMSG	The queue does not contain a message of the desired type.
EFAULT	The pointer to the message is outside the process-allocated address space.
EINTR	The system call was interrupted by an UNIX signal.
EIO	An I/O error occurred during a read or write operation
ENXIO	The requested service cannot be performed on this particular subdevice.
ENOLINK	The link to a requested machine is no longer active.
ESRCH	No process or process group can be found corresponding to the specified PID.
EPERM	The user ID of the sending process is not privileged, and its real or effective user ID does not match the real or saved user ID of the receiving process. The calling process is not sending SIGCONT to a process that shares the same session ID.

Possible CASL values for `errno` are as follows.

Value	Meaning
CA_ERR_ACCESS	The process calling `ca_get_msg()` is not registered. Call `ca_register()` before calling this function.
CA_ERR_IBLK_PTR	The pointer to the `I_Block` is `0` or `-1`.
CA_ERR_INVALID_MAXSZ	The maximum size specified is too large or is `0`.

SEE ALSO

`ca_cancel_def()`, `ca_put_msg()`, `ca_put_msg_def()`, `ca_restart_timer()`, `ca_get_msu_noxudt_t()`

ca_put_cmd()

SYNOPSIS

```
int     ca_put_cmd(
          ipc_key_t   destn_key,
          U32         ref_nbr,
          U16         len,
          U8          *pcmd);
```

DESCRIPTION

The `ca_put_cmd()` function sends an MML command to a specified destination process via the destination's IPC. To receive commands in this manner, the destination process must have registered with the `cmd_allow` parameter set to `1`. This function also sets up the `I_Block` and determines whether the command is longer than the maximum length allowed.

PARAMETERS

* `destn_key` (input)
  Specifies an IPC key that indicates the destination process to which the command is being sent. To use the `ca_put_cmd()` function, you must assign values to the fields in the `ipc_key_t` structure, which is described in the following section, "The IPC Key Structure (`ipc_key_t`)."

* `ref_nbr` (input)
  Specifies a reference number that allows the sending and receiving processes to keep track of messages that are of the same type. The reference number lets the receiving process associate a reply with an instance of a command.

* `len` (input)
  Specifies the length of the command to be sent.

* `pcmd` (input)
  Specifies a pointer to the command to be sent.

IPC Key Structure (`ipc_key_t`)

The following fields make up the `ipc_key_t` structure, which is defined in the include file `sinap.h`.

```
typedef struct ipc_key_s
{
          U32       node;
          U32       module;
          U32       appl;
          U32       proc;
          U8        instance;
          U8        node_index;
          U16       ipc_index;
} ipc_key_t;
```

* `node` (output)
  Specifies the ID of the SINAP node on which your application is running. You can determine this value from the `NODE=` entry in the `/etc/sinap_master` file. You should modify any script files or user-defined program files that contain an invalid, hard-coded node name.

* `module` (output)
  Specifies the name or ID of the module. You can determine this value from the `MODULE=` entry in the `/etc/sinap_master` file. You should modify any script files or user-defined program files that contain an invalid, hard-coded module name.

* `appl` (input)
  Specifies the compressed application ID.

* `proc` (input)
  Specifies the compressed process ID.

* `instance` (input)
  Specifies the instance ID (in the range `1` to `8`). A value of `0` indicates that the field is not used.

* `node_index` (input)
  Specifies the index (0 through 3) of the node.

* `ipc_index` (input)
  Specifies the index ID of the IPC process table.

FILES

`arch.h`, `ca_error.h`, `sinap.h`

RETURN VALUES

The `ca_put_cmd()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

Possible CASL values for `errno` are as follows.

Value	Meaning
`CA_ERR_ACCESS`	The calling process is not registered. Call `ca_register()` before calling this function.
`CA_ERR_DESTN_KEY`	Destination process key not found.
`CA_ERR_CMDS`	Destination process is not registered to receive IPC commands.
`CA_ERR_IBLK_DATA`	The `I_Block` data exceeds the maximum limit.

This function performs a `ca_put_msg()` and can also return the errors listed under that function.

SEE ALSO

`ca_put_reply()`

ca_put_msg()

SYNOPSIS

```
int    ca_put_msg(
         i_block_t   *piblk,
         S8      retry_count);
```

DESCRIPTION

The `ca_put_msg()` function sends a message to a specified destination by means of the IPC queue. If the queue is full, the function returns an error containing the UNIX value `EAGAIN`.

To help prevent messages from being lost when a queue overflows, the failure of a call to `ca_put_msg()` triggers the event `CA_IPC_FAILED`. This event causes the SINAP/SS7 system to generate an alarm and to then call the function `ca_ipc_failed_event()` in order to deliver the alarm to trouble management. By default, the event `CA_IPC_FAILED` causes the SINAP/SS7 system to generate a critical alarm; however, in the trouble-treatment table (`treat.tab`), you can change the alarm's status to minor or you can change the alarm to a notification. (For information on how to do this, see the *SINAP/SS7 User's Guide* (R8051).)

The alarm that the SINAP/SS7 system sends to trouble management contains the following information.

- The type of IPC message that could not be delivered, along with the process that sent the message (the source) and the process to which the message was destined (the destination).
- The value of *`errno`* returned by the `ca_put_msg()` function call that failed.
- The number of times that `ca_ipc_fail_event()` was called by the process attempting to send the IPC message. (This count is important because critical alarms may be lost when a queue overflows.)

PARAMETERS

* `piblk` (input)

Specifies a pointer to a particular `I_Block`. For the `ca_put_msg()` function to work, you must set the following fields in the `i_block_t`, `ipc_trans_t`, and `ipc_data_t` structures, which are described in the sections that follow.

```
i_block_t:dest_id
ipc_trans_t:msg_type
```

```
ipc_trans_t:ref_nbr
ipc_data_t:more_ind
ipc_data_t:len
ipc_data_t:ret_code
```

NOTE

It is assumed that the data portion follows the message field of `i_block_t`. The structure of the data portion is dependent on the value specified in the `msg_type` field.

* `retry_count` (input)

Specifies the number of times `ca_put_msg()` is to resend the message when it encounters a full IPC queue or a system interrupt (signal).

To ensure IPC message delivery during periods of heavy system load, you can define the environment variable, `GUARANTEED_IPC` (no value need be assigned). Defining this variable changes the `retry_count` parameter as follows:

- If `retry_count` is 0, the IPC message is considered to be non-critical and the SINAP/SS7 system does not generate a critical alarm if it cannot deliver the message. Instead, the SINAP/SS7 system returns an error to the user with `errno` typically set to `EAGAIN`.
- If `retry_count` is any value greater than 0, the IPC message is considered critical and every attempt is made to deliver the message, including restarting the SINAP node if necessary. The `ca_put_msg()` function call generates a critical alarm and returns an error only if the message cannot be delivered because of an error condition other than `EAGAIN` (for example, if the called process no longer exists).

Main `I_Block` Structure (`i_block_t`)

The following fields are set in the `i_block_t` structure, which is defined in the include file `iblock.h`. The `iblock.h` include file defines the structure of messages (`I_Blocks`) sent by means of IPC. An `I_Block` is composed of a CASL control part, a transaction part, a timestamp, a node ID, an originator key, a destination key, and a message body.

```
typedef  struct  i_block_s
{
    ca_ctrl_t       ca_ctrl;
    ipc_trans_t     trans;
    timestamp_t     ts;
    node_id_t       node;
    ipc_key_t       orig_id;
    ipc_key_t       dest_id;
    ipc_data_t      msg;
} i_block_t;
```

* `ca_ctrl` (input)
  Specifies the CASL control structure for this `I_Block`. For information about this structure, see "The CASL Control Structure (`ca_ctrl_t`)" later in this section.

* `trans` (input)
  Specifies the transaction ID structure. For information about this structure, see "The `I_Block` Transaction ID Structure (`ipc_trans_t`)" later in this section.

* `ts` (input)
  Specifies a collection of timestamps that the SINAP/SS7 system automatically inserts. The timestamps aid monitoring and logging, and are visible when you run the BITE log-analysis program. For information about this structure's fields, see "The Timestamp Structure (`timestamp_t`)" later in this section.

* `node` (input)
  Specifies the SINAP node ID. This structure is internal to the SINAP/SS7 system and should not be modified.

* `orig_id` (input)
  Specifies the `ipc_key_t` structure that contains the IPC key for a sender application process. For information about the `ipc_key_t` structure, see "The IPC Key Structure (`ipc_key_t`)" later in this section.

* `dest_id` (input)
  Specifies the `ipc_key_t` structure that contains the IPC key for the intended destination process of the `I_Block`. You can obtain this IPC key by calling the `ca_get_key()` function. For information about the `ipc_key_t` structure, see "The IPC Key Structure (`ipc_key_t`)" later in this section.

* `msg` (input)
  Specifies the `ipc_data_t` structure that contains the IPC user data. (For information about the `ipc_data_t` structure, see "The IPC Data Structure (`ipc_data_t`)" later in this section.)

CASL Control Structure (`ca_ctrl_t`)

The following fields make up the `ca_ctrl_t` structure, which is defined in the include file `blkhdr.h`.

```
typedef struct   ca_ctrl_s
{
       int     msg_type;       /* For compatibility with the existing UNIX IPC
                                  mechanism. */
       int     msu_cnt;        /* # of MSUs pending */
       int     free_cnt;       /* # of free MSUs in read queue */
       int     wfree_cnt;      /* # of free MSUs in write queue */
       S16     lost_cnt;       /* # of MSUs lost due to insuff. resources */
       S16     data_size;      /* Total size of structure excluding this
                                  structure. */
       U8      node_index;     /* index (0 - 3) of current node */
       U8      sinap_variant;  /* V_CCITT, V_ANSI, V_HYBRID, V_TTC */
       U16     link;           /* index of the origination link */
       pid_t   pid;            /* Process ID of a specific process or 0
                                  for load distribution */
       int     msg_sender;     /* Set to 0 if from link otherwise contains
                                  the process ID */
       U8      iblk;           /* TRUE if data contains I_Block */
                               /* Not used anywhere!!!!! */
       U8      rw;             /* Flag for monitor message only */
       U8      monitor_id;     /* monitor ID for monitored MSU  */
       U8      ssn_sio;        /* SSN or SIO ID for load distribution. */
       struct {
               U32     timer_id;       /* For CASL internal use only */
               U32     timer_val;      /* For CASL internal use only */
               int     omsg_type;      /* For CASL internal use only */
       } timr;
       struct {                        /* For internal use only */
               int     source;         /* For distribution managment. */
               int     destination;    /* For protocol processing. */
#ifdef _KERNEL
               mblk_t  *mptr;          /* Back pointer to mblk_t.  L3 only.*/
#else
               void    *mptr;          /* ss7-1102 - dummy back pointer. */
#ifdef _LP_32_64_
               U32  filler;            /* For User32/Driver64 compatibility*/
#endif  /* _LP_32_64_ */
#endif  /* _KERNEL */
       } internal;
} ca_ctrl_t;
```

* `msg_type` (output)
  Specifies the type of MSU being sent. This field is compatible with the existing UNIX IPC mechanism.

* `data_size` (output)
  Specifies the total size of the structure, excluding this field.

* `node_index` (output)
  This is an internal field that is automatically initialized to the appropriate value for the SINAP node.

* `sinap_variant` (output)
  This is an internal field that is automatically initialized to the appropriate value for the network variant being used on the SINAP node.

* `lost_cnt` (output)
  Specifies the number of `M_Blocks` lost due to insufficient resources within the SS7 driver.

* `msu_cnt` (output)
  Specifies the number of MSUs pending.

* `free_cnt` (output)
  Specifies the number of free MSUs pending in the read queue.

* `wfree_cnt` (output)
  Specifies the number of free MSUs pending in the write queue.

The following fields are internal to the SINAP/SS7 system and you should not modify them:

* `pid` (output)
* `link` (output)
* `msg_sender` (output)
* `iblk` (output)
* `rw` (output)
* `monitor_id` (output)
* `ssn_sio` (output)
* `source` (output)
* `destination` (output)
* `mptr` (input) Specifies a pointer to `m_block_t`, Level 3.
* `timer_id` (output)
* `timer_val` (output)
* `omsg_type` (output)

IPC Transaction ID Structure (`ipc_trans_t`)

The following fields make up the `ipc_trans_t` structure, which is defined in the include file `iblock.h`.

```
typedef  struct  ipc_trans_s
{
         int     msg_type;
         U32     ref_nbr;
         U16     rw_ind;
         U8      monitor_id;
         U8      scenario_id;
} ipc_trans_t;
```

* `msg_type` (input)
  Specifies the basic message function identifier that the SINAP/SS7 system and client applications use to identify a message. When defining client application messages, you should specify message types within the range of `CL_IPC_MIN` and `CL_IPC_MAX` (see the include file `iblock.h` for more information).

* `ref_nbr` (input)
  Specifies a reference number that allows the sending and receiving processes to keep track of messages that are of the same type. The reference number lets the receiving process associate a reply with an instance of a command.

* `rw_ind` (input)
  Specifies a read or write indicator for the monitor. This field is internal to the SINAP/SS7 system; you should not modify it.

* `monitor_id` (input)
  Associates the IPC message with a particular BITE monitor session. This field is internal to the SINAP/SS7 system; you should not modify it.

* `scenario_id` (input)
  Associates the IPC message with a particular BITE scenario execution session. This field is internal to the SINAP/SS7 system; you should not modify it.

Timestamp Structure (`timestamp_t`)

The `timestamp_t` structure contains the following fields and is defined in the include file `timestamp.h`.

```
typedef struct timestamp_s
{
         U16     index;
         stamp_t stamp[MAX_TIME_STAMPS];
}timestamp_t;
```

* `index` (input)
  Specifies the next slot to stamp the time.

* `stamp[MAX_TIME_STAMPS]` (input)
  Specifies the timestamp slots. See "The `stamp_t` Structure" below for an explanation. (`MAX_TIME_STAMPS` is defined in the SINAP/SS7 `timestamp.h` include file.)

The `stamp_t` Structure

The `stamp_t` structure contains the following fields and is defined in the include file `timestamp.h`.

```
typedef struct stamp_s
{
       U32     secs;                   /* time in seconds since 1/1/70 */
       U8      tsid;                   /* timestamp id                 */
       U8      ipcx;                   /* ipc index if applicable      */
       U16     msec;                   /* time in milliseconds         */
} stamp_t;
```

* `secs` (input)
  Specifies the time (in seconds) since 1/1/70.

* `tsid` (input)
  Specifies the timestamp ID. Valid values are defined in the include file `timestamp.h`.

* `ipcx` (input)
  Specifies the IPC index, if applicable.

* `msec` (input)
  Specifies the time, in milliseconds.

The node_id_t Structure

The node_id_t structure contains the following fields and is defined in the include file iblock.h.

```
typedef struct node_id_s
{
        U8     ni;
        U32    spc;
} node_id_t;
```

* ni (input)
  Specifies the network indicator of the node. This field is internal to the SINAP/SS7 system and should not be modified.

* spc (input)
  Specifies the signaling point code of the node. This field is internal to the SINAP/SS7 system and should not be modified.

IPC Key Structure (ipc_key_t)

The following fields make up the ipc_key_t structure, which is defined in the include file sinap.h.

```
typedef struct ipc_key_s
{
          U32      node;
          U32      module;
          U32      appl;
          U32      proc;
          U8       instance;
          U8       node_index;
          U16      ipc_index;
} ipc_key_t;
```

* node (output)
  Specifies the ID of the SINAP node on which your application is running. You can determine this value from the NODE= entry in the /etc/sinap_master file. You should modify any script files or user-defined program files that contain an invalid, hard-coded node name.

* `module` (output)
  Specifies the name or ID of the module. You can determine this value from the `MODULE=` entry in the `/etc/sinap_master` file. You should modify any script files or user-defined program files that contain an invalid, hard-coded module name.

* `appl` (input)
  Specifies the compressed application ID.

* `proc` (input)
  Specifies the compressed process ID.

* `instance` (input)
  Specifies the instance ID (in the range `1` to `8`). A value of `0` indicates that the field is not used.

* `node_index` (input)
  Specifies the index (0 through3) of the node.

* `ipc_index` (input)
  Specifies the index ID of the IPC process table.

IPC Data Structure (`ipc_data_t`)

The following fields make up the `ipc_data_t` structure, which is defined in the include file `iblock.h`.

```
typedef  struct   ipc_data_s
{
         U8      more_ind;
         U32     len;
         U32     ret_code;
#if defined(__LP64__) || defined(_LP_32_64_)
         U32     filler; /* For User32/Driver64 compatibility */
#endif /* __LP64__ || _LP_32_64_ */
} ipc_data_t;
```

* `more_ind` (input)
  Specifies whether an IPC message is the last in a sequence. This field is useful if the data portion of a message exceeds the maximum amount of data that UNIX can send in a single data packet. Note that the limit is an UNIX configuration parameter, initially set to 4096 octets. The SINAP/SS7 system also uses `more_ind` when a command reply exceeds the response timeout. In this case, a command reply could consist of an arbitrary number of messages and a final reply; the `more_ind` field would be set to `1` to indicate that the receiving process is working on the reply. The final reply would indicate the result of the command.

* `len` (input)
  Specifies the length (in octets) of the data portion of the message body.

* `ret_code` (input)
  Specifies a return code value. By returning a user-defined value, a client application can use this field to indicate success or failure.

NOTE

The data portion of a message should follow the message field of `i_block_t`. The structure of the data portion is dependent on the `msg_type` field. The following use of `i_block_t` is recommended.

```
typedef struct user_struc_s
{
  i_block_t   iblk_hdr;
  char        user_data[MAX_IBLK_DATA_SZ];
} user_struc_t;
```

FILES

`arch.h`, `ca_error.h`, `iblock.h`, `sinap.h`, `sysdefs.h`, `sys/time.h`, `timestamp.h`

RETURN VALUES

The `ca_put_msg()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

Possible UNIX values for `errno` are as follows.

Value	Meaning
EBADF	An invalid open file descriptor was specified.
ENOTTY	This `fides` is not associated with a device driver that accepts control functions.
EINVAL	Queue ID is not a valid message queue ID. The value of `msg_type` is less than `1`, or `msg_sz` is greater than `0` or the system-imposed limit.
EACCES	Operation permission is denied to the calling process.
EAGAIN	The queue is full.
EFAULT	The pointer to the specified message is outside the address space allocated to the process.
EINTR	The system call was interrupted by an UNIX signal.

Possible CASL values for `errno` are as follows.

Value	Meaning
CA_ERR_ACCESS	The process calling `ca_put_msg()` is not registered. Call `ca_register()` before calling this function.
CA_ERR_DESTN_KEY	The IPC key specified in the `dest_id` field of the `i_block_t` structure could not be found.
CA_ERR_IBLK_DATA	The `I_Block` data exceeds the maximum limit.
CA_ERR_IBLK_MSGTYPE	Invalid message type.

This function performs a `ca_get_key()` and can also return the errors listed under that function.

SEE ALSO

`ca_get_msg()`, `ca_put_msg_def()`

ca_put_msg_def()

SYNOPSIS

```
int     ca_put_msg_def(
          U32         timer_id,
          int         timer_val,
          i_block_t   *piblk);
```

DESCRIPTION

The `ca_put_msg_def()` function sends a deferred message to a specified destination by means of the IPC queue.

To help prevent messages from being lost when a queue overflows, the failure of a call to `ca_put_msg_def()` triggers the event `CA_IPC_FAILED`. This event causes the SINAP/SS7 system to generate an alarm and to then call the function `ca_ipc_failed_event()` in order to deliver the alarm to trouble management. By default, the event `CA_IPC_FAILED` causes the SINAP/SS7 system to generate a critical alarm; however, in the trouble-treatment table (`treat.tab`), you can change the alarm's status to minor or you can change the alarm to a notification. (For information on how to do this, see the *SINAP/SS7 User's Guide* (R8051).)

The alarm that the SINAP/SS7 system sends to trouble management contains the following information.

- The type of IPC message that could not be delivered, along with the process that sent the message (the source) and the process to which the message was destined (the destination).
- The value of `errno` returned by the `ca_put_msg_def()` function call that failed.
- The number of times that `ca_ipc_fail_event()` was called by the process attempting to send the IPC message. (This count is important because critical alarms may be lost when a queue overflows.)

NOTES

1. Deferred messages are handled by the SINAP Management process Deferred Message Handler (NMDM), which can store up to 4096 deferred IPC messages, each to be delivered when its timeout occurs after `timer_val`

milliseconds. The resolution of NMDM is 100 milliseconds (1/10 second).

2. A `CA_IPC_FAILED_EVENT` alarm can happen in one of three ways:
   - NMDM IPC input queue is full. Call to `ca_put_msg_def()` returns error with errno of `EAGAIN (11)`, and an alarm is sent with the above information, where the source process is the caller of `ca_put_msg_def()`, the destination process is NMDM, and errno is `EAGAIN`.
   - The deferred message's destination process IPC input queue is full. Call to `ca_put_msg_def()` will have already succeeded. An alarm is sent with the above information, where the source process is NMDM, the destination process is that which was specified by the `i_block_tdest_id` field, the errno is `EAGAIN`.
   - The deferred message could not be stored by NMDM as its limit of 4096 deferred messages has been reached. Call to `ca_put_msg_def()` will have already succeeded. An alarm is sent with the above information, where the source process is NMDM, the destination process is that which was specified by the `i_block_t dest_id` field, and errno is `ENOMEM (12)`.

PARAMETERS

* `timer_id` (input)
  Specifies a unique value that identifies the message. A process other than the originating process within the same application may cancel deferred message delivery. Timer IDs must be unique to each client application. However, different applications can use the same timer ID.

  To have the `ca_put_msg_def()` function create a unique timer ID, specify a value of `0` for this parameter. In this case, the function creates a unique timer ID and returns it. This is useful when a client process requires a unique timer ID for a particular instance of a process.

* `timer_val` (input)
  Specifies the time (in milliseconds) to delay the message.

* `piblk` (input)
  Specifies a pointer to a particular `I_Block`. For the `ca_put_msg_def()` function to work, you must set the following fields in the `i_block_t`, `ipc_trans_t`, and

`ipc_data_t` structures. For an explanation of these fields and possible values, see the following section, "The Main `I_Block` Structure (`i_block_t`)."

```
i_block_t:dest_id
ipc_trans_t:msg_type
ipc_trans_t:ref_nbr
ipc_data_t:more_ind
ipc_data_t:len
ipc_data_t:ret_code
```

Main `I_Block` Structure (`i_block_t`)

The following fields are set in the `i_block_t` structure, which is defined in the include file `iblock.h`. The `iblock.h` include file defines the structure of messages (`I_Blocks`) sent via IPC. An `I_Block` is composed of a CASL control part, a transaction part, a timestamp, a node ID, an originator key, a destination key, and a message body.

```
typedef  struct  i_block_s
{
    ca_ctrl_t       ca_ctrl;
    ipc_trans_t     trans;
    timestamp_t     ts;
    node_id_t       node;
    ipc_key_t       orig_id;
    ipc_key_t       dest_id;
    ipc_data_t      msg;
} i_block_t;
```

* `ca_ctrl` (input)
  Specifies the CASL control structure for this `I_Block`. For more information about this structure, see "The CASL Control Structure (`ca_ctrl_t`)" later in this section.

* `trans` (input)
  Specifies the transaction ID structure. For information about this structure, see "The `I_Block` Transaction ID Structure (`ipc_trans_t`)" later in this section.

* `ts` (input)
  Specifies a collection of timestamps that the SINAP/SS7 system automatically inserts. The timestamps aid monitoring and logging, and are visible when you run the BITE log-analysis program. For information about this structure's fields, see "The Timestamp Structure (`timestamp_t`)" later in this section.

* `node` (input)
  Specifies the node ID. This structure is internal to the SINAP/SS7 system and should not be modified.

* `orig_id` (input)
  Specifies the `ipc_key_t` structure that contains the IPC key for a sender application process. For information about the `ipc_key_t` structure, see "The IPC Key Structure (`ipc_key_t`)" later in this section.

* `dest_id` (input)
  Specifies the `ipc_key_t` structure that contains the IPC key for the intended destination process of the `I_Block`. You can obtain this IPC key by calling the `ca_get_key()` function. For information about the `ipc_key_t` structure, see "The IPC Key Structure (`ipc_key_t`)" later in this section.

* `msg` (input)
  Specifies the `ipc_data_t` structure that contains the IPC user data. For information about the `ipc_data_t` structure, see "The IPC Data Structure (`ipc_data_t`)" later in this section.

CASL Control Structure (`ca_ctrl_t`)

The following fields make up the `ca_ctrl_t` structure, which is defined in the include file `blkhdr.h`.

```
typedef struct   ca_ctrl_s
{
       int     msg_type;       /* For compatibility with the existing UNIX IPC
                                     mechanism. */
       int     msu_cnt;        /* # of MSUs pending */
       int     free_cnt;       /* # of free MSUs in read queue */
       int     wfree_cnt;      /* # of free MSUs in write queue */
       S16     lost_cnt;       /* # of MSUs lost due to insuff. resources */
       S16     data_size;      /* Total size of structure excluding this
                                  structure. */
       U8      node_index;     /* index (0 - 3) of current node */
       U8      sinap_variant;  /* V_CCITT, V_ANSI, V_HYBRID, V_TTC */
       U16     link;           /* index of the origination link */
       pid_t   pid;            /* Process ID of a specific process or 0
                                  for load distribution */
       int     msg_sender;     /* Set to 0 if from link otherwise contains
                                  the process ID */
       U8      iblk;           /* TRUE if data contains I_Block */
                               /* Not used anywhere!!!!! */
       U8      rw;             /* Flag for monitor message only */
       U8      monitor_id;     /* monitor ID for monitored MSU  */
       U8      ssn_sio;        /* SSN or SIO ID for load distribution. */
       struct {
               U32     timer_id;       /* For CASL internal use only */
               U32     timer_val;      /* For CASL internal use only */
               int     omsg_type;      /* For CASL internal use only */
       } timr;
       struct {                        /* For internal use only */
               int     source;         /* For distribution managment. */
               int     destination;    /* For protocol processing. */
#ifdef _KERNEL
               mblk_t  *mptr;          /* Back pointer to mblk_t.  L3 only.*/
#else
               void    *mptr;          /* ss7-1102 - dummy back pointer. */
#ifdef _LP_32_64_
               U32  filler;            /* For User32/Driver64 compatibility*/
#endif  /* _LP_32_64_ */
#endif  /* _KERNEL */
       } internal;
} ca_ctrl_t;
```

* `msg_type` (output)
  Specifies the type of MSU being sent. This field is compatible with the existing UNIX IPC mechanism.

* `data_size` (output)
  Specifies the total size of the structure, excluding this field.

* `node_index` (output)
  This is an internal field that is automatically initialized to the appropriate value for the SINAP node.

* `sinap_variant` (output)
  This is an internal field that is automatically initialized to the appropriate value for the network variant being used on the SINAP node.

* `lost_cnt` (output)
  Specifies the number of `M_Blocks` lost due to insufficient resources within the SS7 driver.

* `msu_cnt` (output)
  Specifies the number of MSUs pending.

* `free_cnt` (output)
  Specifies the number of free MSUs pending in the read queue.

* `wfree_cnt` (output)
  Specifies the number of free MSUs pending in the write queue.

The following fields are internal to the SINAP/SS7 system and you should not modify them:

* `pid` (output)
* `link` (output)
* `msg_sender` (output)
* `iblk` (output)
* `rw` (output)
* `monitor_id` (output)
* `ssn_sio` (output)
* `source` (output)
* `destination` (output)
* `mptr` (input) Specifies a pointer to `m_block_t`, Level 3.
* `timer_id` (output)
* `timer_val` (output)
* `omsg_type` (output)

IPC Transaction ID Structure (`ipc_trans_t`)

The following fields make up the `ipc_trans_t` structure, which is defined in the include file `iblock.h`.

```
typedef  struct  ipc_trans_s
{
         int     msg_type;
         U32     ref_nbr;
         U16     rw_ind;
         U8      monitor_id;
         U8      scenario_id;
} ipc_trans_t;
```

* `msg_type` (input)
  Specifies the basic message function identifier that the SINAP/SS7 system and client applications use to identify a message. When defining client application messages, you should specify message types within the range of `CL_IPC_MIN` and `CL_IPC_MAX` (see the include file `iblock.h` for more information).

* `ref_nbr` (input)
  Specifies a reference number that allows the sending and receiving processes to keep track of messages that are of the same type. The reference number lets the receiving process associate a reply with an instance of a command.

* `rw_ind` (input)
  Specifies a read or write indicator for the monitor. This field is internal to the SINAP/SS7 system; you should not modify it.

* `monitor_id` (input)
  Associates the IPC message with a particular BITE monitor session. This field is internal to the SINAP/SS7 system; you should not modify it.

* `scenario_id` (input)
  Associates the IPC message with a particular BITE scenario execution session. This field is internal to the SINAP/SS7 system; you should not modify it.

Timestamp Structure (`timestamp_t`)

The `timestamp_t` structure contains the following fields and is defined in the include file `timestamp.h`.

```
typedef struct timestamp_s
{
         U16     index;
         stamp_t stamp[MAX_TIME_STAMPS];
}timestamp_t;
```

* `index` (input)
  Specifies the next slot to stamp the time.

* `stamp[MAX_TIME_STAMPS]` (input)
  Specifies the timestamp slots. See "The `stamp_t` Structure" below for an explanation. (`MAX_TIME_STAMPS` is defined in the SINAP/SS7 `timestamp.h` include file.)

The `stamp_t` Structure

The `stamp_t` structure contains the following fields and is defined in the include file `timestamp.h`.

```
typedef struct stamp_s
{
      U32     secs;                  /* time in seconds since 1/1/70 */
      U8      tsid;                  /* timestamp id                 */
      U8      ipcx;                  /* ipc index if applicable      */
      U16     msec;                  /* time in milliseconds         */
} stamp_t;
```

* `secs` (input)
  Specifies the time (in seconds) since 1/1/70.

* `tsid` (input)
  Specifies the timestamp ID. Valid values are defined in the include file `timestamp.h`.

* `ipcx` (input)
  Specifies the IPC index, if applicable.

* `msec` (input)
  Specifies the time, in milliseconds.

The node_id_t Structure

The node_id_t structure contains the following fields and is defined in the include file iblock.h.

```
typedef struct node_id_s
{
        U8     ni;
        U32    spc;
} node_id_t;
```

* ni (input)
  Specifies the network indicator of the node. This field is internal to the SINAP/SS7 system and should not be modified.

* spc (input)
  Specifies the signaling point code of the node. This field is internal to the SINAP/SS7 system and should not be modified.

IPC Key Structure (ipc_key_t)

The ipc_key_t structure contains the following fields and is defined in the include file sinap.h.

```
typedef struct ipc_key_s
{
          U32       node;
          U32       module;
          U32       appl;
          U32       proc;
          U8        instance;
          U8        node_index;
          U16       ipc_index;
} ipc_key_t;
```

* node (output)
  Specifies the ID of the SINAP node on which your application is running. You can determine this value from the NODE= entry in the /etc/sinap_master file. You should modify any script files or user-defined program files that contain an invalid, hard-coded node name.

* `module` (output)
  Specifies the name or ID of the module. You can determine this value from the `MODULE=` entry in the `/etc/sinap_master` file. You should modify any script files or user-defined program files that contain an invalid, hard-coded module name.

* `appl` (input)
  Specifies the compressed application ID.

* `proc` (input)
  Specifies the compressed process ID.

* `instance` (input)
  Specifies the instance ID (in the range `1` to `8`). A value of `0` indicates that the field is not used.

* `node_index` (input)
  Specifies the index (0 through 3) of the node.

* `ipc_index` (input)
  Specifies the index ID of the IPC process table.

IPC Data Structure (`ipc_data_t`)

The following fields make up the `ipc_data_t` structure, which is defined in the include file `iblock.h`.

```
typedef  struct   ipc_data_s
{
         U8      more_ind;
         U32     len;
         U32     ret_code;
#if defined(__LP64__) || defined(_LP_32_64_)
         U32     filler; /* For User32/Driver64 compatibility */
#endif /* __LP64__ || _LP_32_64_ */
} ipc_data_t;
```

* `more_ind` (input)
  Specifies whether an IPC message is the last in a sequence. This field is useful if the data portion of a message exceeds the maximum amount of data that the UNIX operating system can send in a single data packet. Note that the limit is a UNIX configuration parameter, initially set to 4096 octets. The SINAP/SS7 system also uses `more_ind` when a command reply exceeds the response timeout. In this case, a command reply could consist of an arbitrary number of messages and a final reply; the `more_ind` field would be set to `1` to indicate that the receiving process is working on the reply. The final reply would indicate the result of the command.

* `len` (input)
  Specifies the length (in octets) of the data portion of the message body.

* `ret_code` (input)
  Specifies a return code value. By returning a user-defined value, a client application can use this field to indicate success or failure.

NOTE

The data portion of a message should follow the message field of `i_block_t`. The structure of the data portion is dependent on the `msg_type` field. The following use of `i_block_t` is recommended.

```
typedef struct user_struc_s
{
  i_block_t   iblk_hdr;
  char        user_data[MAX_IBLK_DATA_SZ];
} user_struc_t;
```

FILES

`arch.h`, `ca_error.h`, `iblock.h`, `sinap.h`, `sysdefs.h`, `sys/time.h`, `timestamp.h`

RETURN VALUES

The `ca_put_msg_def()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful.
`timer ID`	CASL or user-provided timer ID.

Possible CASL values for `errno` are as follows.

Value	Meaning
`CA_ERR_ACCESS`	The process calling `ca_put_msg_def()` is not registered. Call `ca_register()` before calling this function.
`CA_ERR_DESTN_KEY`	The IPC key specified in the `dest_id` field of the `i_block_t` structure could not be found.
`CA_ERR_IBLK_DATA`	The `I_Block` data exceeds the maximum limit.
`CA_ERR_IBLK_MSGTYPE`	Invalid message type.

This function performs a `ca_get_key()` and can also return the errors listed under that function.

SEE ALSO

`ca_cancel_def()`, `ca_get_msg()`, `ca_put_msg()`, `ca_restart_timer()`

ca_put_reply()

SYNOPSIS

```
int    ca_put_reply(
         ipc_key_t   destn_key,
         U32         ref_nbr,
         BOOL        fmore,
         U16         len,
         U8          *preply);
```

DESCRIPTION

The `ca_put_reply()` function sends an IPC message reply to the specified destination. This function is used to respond to a command sent by the `ca_put_cmd()` function.

PARAMETERS

* `destn_key` (input)
  Specifies the IPC key of the reply's destination. Use the value of `orig_id` returned by the call to `ca_get_msg()`. (The IPC key is defined by the `ipc_key_t` structure, which is defined in the include file `sinap.h`. For an explanation of this structure's fields, see the following section, "The IPC Key Structure (`ipc_key_t`).")

* `ref_nbr` (input)
  Specifies a reference number that allows the sending and receiving processes to keep track of messages that are of the same type. The reference number lets the receiving process associate a reply with an instance of a command.

* `fmore` (input)
  Specifies whether there are more replies. Use `1` to indicate that there are more replies; otherwise, use `0`.

* `len` (input)
  Specifies the length of the reply.

* `preply` (input)
  Specifies a pointer to the reply.

IPC Key Structure (`ipc_key_t`)

The `ipc_key_t` structure contains the following fields and is defined in the include file `sinap.h`.

```
typedef struct ipc_key_s
{
          U32      node;
          U32      module;
          U32      appl;
          U32      proc;
          U8       instance;
          U8       node_index;
          U16      ipc_index;
} ipc_key_t;
```

* `node` (output)
  Specifies the ID of the SINAP node on which your application is running. You can determine this value from the `NODE=` entry in the `/etc/sinap_master` file. You should modify any script files or user-defined program files that contain an invalid, hard-coded node name.

* `module` (output)
  Specifies the name or ID of the module. You can determine this value from the `MODULE=` entry in the `/etc/sinap_master` file. You should modify any script files or user-defined program files that contain an invalid, hard-coded module name.

* `appl` (input)
  Specifies the compressed application ID.

* `proc` (input)
  Specifies the compressed process ID.

* `instance` (input)
  Specifies the instance ID (in the range `1` to `8`). A value of `0` indicates that the field is not used.

* `node_index` (input)
  Specifies the index (0 through 3) of the node.

* `ipc_index` (input)
  Specifies the index ID of the IPC process table.

FILES

`arch.h`, `ca_error.h`, `sinap.h`,

RETURN VALUES

The `ca_put_reply()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful.

Possible CASL values for `errno` are as follows.

Value	Meaning
`CA_ERR_ACCESS`	The process is not registered. Call `ca_register()` before calling this function.
`CA_ERR_DESTN_KEY`	Destination process key not found.
`CA_ERR_IBLK_DATA`	The `I_Block` data exceeds the maximum limit.

This function performs a `ca_put_msg()` and can also return the errors listed under that function.

SEE ALSO

`ca_put_cmd()`

ca_restart_timer()

SYNOPSIS

```
int     restart_timer(
          U32     timer_id,
          int     timer_val);
```

DESCRIPTION

The `ca_restart_timer()` function restarts the timer for all deferred messages associated with the specified timer ID.

PARAMETERS

* `timer_id` (input)
  Specifies a 32-bit timer ID. Use the timer ID returned by the `ca_put_msg_def()` function.

* `timer_val` (input)
  Specifies the time (in milliseconds) to restart the timer that delays the message.

FILES

`arch.h`, `ca_error.h`

RETURN VALUES

The `ca_restart_timer()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

Possible CASL values for `errno` are as follows.

Value	Meaning
CA_ERR_DESTN_KEY	Destination process key not found.
CA_ERR_ACCESS	The process is not registered.

This function performs a `ca_put_msg()` and a `ca_get_msg()` and can also return the errors listed under those functions.

SEE ALSO

`ca_cancel_def()`, `ca_get_msg()`, `ca_put_msg()`, `ca_put_msg_def()`

ca_swap_keys()

SYNOPSIS

```
int     ca_swap_keys(
          i_block_t   *piblk);
```

DESCRIPTION

The `ca_swap_keys()` function swaps the origination (`orig_id`) and destination (`dest_id`) key portions of an `I_Block`.

PARAMETERS

* `piblk` (input)
  Specifies a pointer to the `i_block_t` structure containing the `I_Block`. For a description of this structure's fields, see the following section, "The Main `I_Block` Structure (`i_block_t`)."

Main `I_Block` Structure (`i_block_t`)

The following fields are set in the `i_block_t` structure, which is defined in the include file `iblock.h`. The `iblock.h` include file defines the structure of messages (`I_Blocks`) sent by means of IPC. An `I_Block` is composed of a CASL control part, a transaction part, a timestamp, a node ID, an originator key, a destination key, and a message body.

```
typedef  struct  i_block_s
{
    ca_ctrl_t       ca_ctrl;
    ipc_trans_t     trans;
    timestamp_t     ts;
    node_id_t       node;
    ipc_key_t       orig_id;
    ipc_key_t       dest_id;
    ipc_data_t      msg;
} i_block_t;
```

* `ca_ctrl` (input)
  Specifies the CASL control structure for this `I_Block`. For more information about this structure, see "The CASL Control Structure (`ca_ctrl_t`)" later in this section.

* `trans` (input)
  Specifies the transaction ID structure. For information about this structure, see "The `I_Block` Transaction ID Structure (`ipc_trans_t`)" later in this section.

* `ts` (input)
  Specifies a collection of timestamps that the SINAP/SS7 system automatically inserts. The timestamps aid monitoring and logging, and are visible when you run the BITE log-analysis program. For information about this structure's fields, see "The Timestamp Structure (`timestamp_t`)" later in this section.

* `node` (input)
  Specifies the node ID. This structure is internal to the SINAP/SS7 system and should not be modified.

* `orig_id` (input)
  Specifies the `ipc_key_t` structure that contains the IPC key for a sender application process. For information about the `ipc_key_t` structure, see "The IPC Key Structure (`ipc_key_t`)" later in this section.

* `dest_id` (input)
  Specifies the `ipc_key_t` structure that contains the IPC key for the intended destination process of the `I_Block`. You can obtain this IPC key by calling the `ca_get_key()` function. For information about the `ipc_key_t` structure, see "The IPC Key Structure (`ipc_key_t`)" later in this section.

* `msg` (input)
  Specifies the `ipc_data_t` structure that contains the IPC user data. For information about the `ipc_data_t` structure, see "The IPC Data Structure (`ipc_data_t`)" later in this section.

CASL Control Structure (`ca_ctrl_t`)

The following fields make up the `ca_ctrl_t` structure, which is defined in the include file `blkhdr.h`.

```
typedef struct   ca_ctrl_s
{
       int     msg_type;       /* For compatibility with the existing UNIX IPC
                                     mechanism. */
       int     msu_cnt;        /* # of MSUs pending */
       int     free_cnt;       /* # of free MSUs in read queue */
       int     wfree_cnt;      /* # of free MSUs in write queue */
       S16     lost_cnt;       /* # of MSUs lost due to insuff. resources */
       S16     data_size;      /* Total size of structure excluding this
                                  structure. */
       U8      node_index;     /* index (0 - 3) of current node */
       U8      sinap_variant;  /* V_CCITT, V_ANSI, V_HYBRID, V_TTC */
       U16     link;           /* index of the origination link */
       pid_t   pid;            /* Process ID of a specific process or 0
                                  for load distribution */
       int     msg_sender;     /* Set to 0 if from link otherwise contains
                                  the process ID */
       U8      iblk;           /* TRUE if data contains I_Block */
                               /* Not used anywhere!!!!! */
       U8      rw;             /* Flag for monitor message only */
       U8      monitor_id;     /* monitor ID for monitored MSU  */
       U8      ssn_sio;        /* SSN or SIO ID for load distribution. */
       struct {
               U32     timer_id;       /* For CASL internal use only */
               U32     timer_val;      /* For CASL internal use only */
               int     omsg_type;      /* For CASL internal use only */
       } timr;
       struct {                        /* For internal use only */
               int     source;         /* For distribution managment. */
               int     destination;    /* For protocol processing. */
#ifdef _KERNEL
               mblk_t  *mptr;          /* Back pointer to mblk_t.  L3 only.*/
#else
               void    *mptr;          /* ss7-1102 - dummy back pointer. */
#ifdef _LP_32_64_
               U32  filler;            /* For User32/Driver64 compatibility*/
#endif  /* _LP_32_64_ */
#endif  /* _KERNEL */
       } internal;
} ca_ctrl_t;
```

* `msg_type` (output)
  Specifies the type of MSU being sent. This field is compatible with the existing UNIX IPC mechanism.

* `data_size` (output)
  Specifies the total size of the structure, excluding this field.

* `node_index` (output)
  This is an internal field that is automatically initialized to the appropriate value for the SINAP node.

* `sinap_variant` (output)
  This is an internal field that is automatically initialized to the appropriate value for the network variant being used on the SINAP node.

* `lost_cnt` (output)
  Specifies the number of `M_Blocks` lost due to insufficient resources within the SS7 driver.

* `msu_cnt` (output)
  Specifies the number of MSUs pending.

* `free_cnt` (output)
  Specifies the number of free MSUs pending in the read queue.

* `wfree_cnt` (output)
  Specifies the number of free MSUs pending in the write queue.

The following fields are internal to the SINAP/SS7 system and you should not modify them:

* `pid` (output)
* `link` (output)
* `msg_sender` (output)
* `iblk` (output)
* `rw` (output)
* `monitor_id` (output)
* `ssn_sio` (output)
* `source` (output)
* `destination` (output)
* `mptr` (input) Specifies a pointer to `m_block_t`, Level 3.
* `timer_id` (output)
* `timer_val` (output)
* `omsg_type` (output)

IPC Transaction ID Structure (`ipc_trans_t`)

The following fields make up the `ipc_trans_t` structure, which is defined in the include file `iblock.h`.

```
typedef  struct  ipc_trans_s
{
         long    msg_type;
         U32     ref_nbr;
         U16     rw_ind;
         U8      monitor_id;
         U8      scenario_id;
} ipc_trans_t;
```

* `msg_type` (input)
  Specifies the basic message function identifier that the SINAP/SS7 system and client applications use to identify a message. When defining client application messages, you should specify message types within the range of `CL_IPC_MIN` and `CL_IPC_MAX` (see the include file `iblock.h` for more information).

* `ref_nbr` (input)
  Specifies a reference number that allows the sending and receiving processes to keep track of messages that are of the same type. The reference number lets the receiving process associate a reply with an instance of a command.

* `rw_ind` (input)
  Specifies a read or write indicator for the monitor. This field is internal to the SINAP/SS7 system; you should not modify it.

* `monitor_id` (input)
  Associates the IPC message with a particular BITE monitor session. This field is internal to the SINAP/SS7 system; you should not modify it.

* `scenario_id` (input)
  Associates the IPC message with a particular BITE scenario execution session. This field is internal to the SINAP/SS7 system; you should not modify it.

Timestamp Structure (`timestamp_t`)

The `timestamp_t` structure contains the following fields and is defined in the include file `timestamp.h`.

```
typedef struct timestamp_s
{
         U16     index;
         stamp_t stamp[MAX_TIME_STAMPS];
}timestamp_t;
```

* `index` (input)
  Specifies the next slot to stamp the time.

* `stamp[MAX_TIME_STAMPS]` (input)
  Specifies the timestamp slots. See "The `stamp_t` Structure" below for an explanation. (`MAX_TIME_STAMPS` is defined in the SINAP/SS7 `timestamp.h` include file.)

The `stamp_t` Structure

The `stamp_t` structure contains the following fields and is defined in the include file `timestamp.h`.

```
typedef struct stamp_s
{
       U32     secs;                  /* time in seconds since 1/1/70 */
       U8      tsid;                  /* timestamp id                 */
       U8      ipcx;                  /* ipc index if applicable      */
       U16     msec;                  /* time in milliseconds         */
} stamp_t;
```

* `secs` (input)
  Specifies the time (in seconds) since 1/1/70.

* `tsid` (input)
  Specifies the timestamp ID. Valid values are defined in the include file `timestamp.h`.

* `ipcx` (input)
  Specifies the IPC index, if applicable.

* `msec` (input)
  Specifies the time, in milliseconds.

The `node_id_t` Structure

The `node_id_t` structure contains the following fields and is defined in the include file `iblock.h`.

```
typedef struct node_id_s
{
        U8     ni;
        U32    spc;
} node_id_t;
```

* `ni` (input)
  Specifies the network indicator of the node. This field is internal to the SINAP/SS7 system and should not be modified.

* `spc` (input)
  Specifies the signaling point code of the node. This field is internal to the SINAP/SS7 system and should not be modified.

IPC Key Structure (`ipc_key_t`)

The `ipc_key_t` structure contains the following fields and is defined in the include file `sinap.h`.

```
typedef struct ipc_key_s
{
          U32      node;
          U32      module;
          U32      appl;
          U32      proc;
          U8       instance;
          U8       node_index;
          U16      ipc_index;
} ipc_key_t;
```

* `node` (output)
  Specifies the ID of the SINAP node on which your application is running. You can determine this value from the `NODE=` entry in the `/etc/sinap_master` file. You

should modify any script files or user-defined program files that contain an invalid, hard-coded node name.

* `module` (output)
  Specifies the name or ID of the module. You can determine this value from the `MODULE=` entry in the `/etc/sinap_master` file. You should modify any script files or user-defined program files that contain an invalid, hard-coded module name.

* `appl` (input)
  Specifies the compressed application ID.

* `proc` (input)
  Specifies the compressed process ID.

* `instance` (input)
  Specifies the instance ID (in the range `1` to `8`). A value of `0` indicates that the field is not used.

* `node_index` (input)
  Specifies the index (0 through 3) for the node.

* `ipc_index` (input)
  Specifies the index ID of the IPC process table.

IPC Data Structure (`ipc_data_t`)

The following fields make up the `ipc_data_t` structure, which is defined in the include file `iblock.h`.

```
typedef  struct   ipc_data_s
{
         U8      more_ind;
         U32     len;
         U32     ret_code;
#if defined(__LP64__) || defined(_LP_32_64_)
         U32     filler; /* For User32/Driver64 compatibility */
#endif /* __LP64__ || _LP_32_64_ */
} ipc_data_t;
```

* `more_ind` (input)
  Specifies whether an IPC message is the last in a sequence. This field is useful if the data portion of a message exceeds the maximum amount of data that UNIX can send in a single data packet. Note that the limit is an UNIX configuration parameter, initially set to 4096 octets. The SINAP/SS7 system also uses `more_ind` when a command reply exceeds the response timeout. In this case, a command reply could consist of an arbitrary number of messages and a final reply; the `more_ind` field would be set to `1` to indicate that the

receiving process is working on the reply. The final reply would indicate the result of the command.

* `len` (input)
  Specifies the length (in octets) of the data portion of the message body.

* `ret_code` (input)
  Specifies a return code value. By returning a user-defined value, a client application can use this field to indicate success or failure.

NOTE

The data portion of a message should follow the message field of `i_block_t`. The structure of the data portion is dependent on the `msg_type` field. The following use of `i_block_t` is recommended.

```
typedef struct user_struc_s
{
  i_block_t   iblk_hdr;
  char        user_data[MAX_IBLK_DATA_SZ];
} user_struc_t;
```

FILES

`arch.h`, `ca_error.h`, `iblock.h`, `sinap.h`, `sysdefs.h`, `sys/time.h`, `timestamp.h`

RETURN VALUES

The `ca_swap_keys()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

A possible CASL error follows.

Value	Meaning
`CA_ERR_IPC_KEY`	IPC key is 0.

SEE ALSO

`ca_ascii_u32()`, `ca_check_key()`, `ca_get_key()`, `ca_u32_ascii()`

ca_u32_ascii()

SYNOPSIS

```
int   ca_u32_ascii(
        ipc_key_t   *pipc_key,
        char        *pn_ret,
        char        *pm_ret,
        char        *pa_ret,
        char        *pp_ret);
```

DESCRIPTION

The `ca_u32_ascii()` function converts an IPC key's node, module, application, and process names from U32 words to ASCII strings. The function returns five pointers, all of which contain four-byte ASCII values.

PARAMETERS

* `pipc_key` (input)
  Specifies a pointer to the `ipc_key_t` structure that contains the IPC key whose node, module, application (`appl`), and process (`proc`) names you want to convert from U32 words to ASCII strings. For a description of this structure's fields, see "The IPC Key Structure (`ipc_key_t`)" later in this section.

* `pn_ret` (output)
  Specifies a pointer to the node name of the calling process. The name can be up to four bytes long in an ASCII string.

* `pm_ret` (output)
  Specifies a pointer to the module name of the calling process. The name can be up to four bytes long in an ASCII string.

* `pa_ret` (output)
  Specifies a pointer to the application name of the calling process. This name can be up to four bytes long in an ASCII string.

* `pp_ret` (output)
  Specifies a pointer to the process name of the calling process. This name can be up to four bytes long in an ASCII string.

IPC Key Structure (`ipc_key_t`)

The `ipc_key_t` structure contains the following fields and is defined in the include file `sinap.h`.

```
typedef struct ipc_key_s
{
          U32       node;
          U32       module;
          U32       appl;
          U32       proc;
          U8        instance;
          U8        node_index;
          U16       ipc_index;
} ipc_key_t;
```

* `node` (output)
  Specifies the ID of the SINAP node on which your application is running. You can determine this value from the `NODE=` entry in the `/etc/sinap_master` file. You should modify any script files or user-defined program files that contain an invalid, hard-coded node name.

* `module` (output)
  Specifies the name or ID of the module. You can determine this value from the `MODULE=` entry in the `/etc/sinap_master` file. You should modify any script files or user-defined program files that contain an invalid, hard-coded module name.

* `appl` (input)
  Specifies the compressed application ID.

* `proc` (input)
  Specifies the compressed process ID.

* `instance` (input)
  Specifies the instance ID (in the range `1` to `8`). A value of `0` indicates that the field is not used.

* `node_index` (input)
  Specifies the index (0 through 3) of the node.

* `ipc_index` (input)
  Specifies the index ID of the IPC process table.

FILES

`arch.h`, `ca_error.h`, `sinap.h`

RETURN VALUES

The `ca_u32_ascii()` function returns 0 if successful.

SEE ALSO

`ca_ascii_u32()`, `ca_check_key()`, `ca_get_key()`, `ca_swap_keys()`

Load Control Functions

This section contains an alphabetic reference of the following CASL functions, which you can include in an application in order to implement the load control facility.

- `ca_disable_locon()` terminates load control operation.
- `ca_enable_locon()` initiates load control operation.
- `ca_exit_locon()` deactivates load control processing.
- `ca_inquire_locon()` retrieves load control statistics.
- `ca_invoke_locon()` causes the SINAP/SS7 system to begin performing load control processing.
- `ca_setup_locon()` configures an application for load control and defines load control operating characteristics.

> NOTE
>
> The include file `$SINAP_HOME/Include/locon.h` contains definitions for the load control functions.

The following two sections provide instructions for implementing load control in an application. See the section “Load Control” in Chapter 3 for information on items you should be aware of as you implement load control for an application.

Implementing Load Control in an Application

Only applications that register with the SINAP/SS7 system at the TCAP boundary can implement load control functionality. For information about how an application registers with the SINAP/SS7 system, see the description of the `ca_register()` function earlier in this chapter.

Typically, an application's control process implements the functions for enabling, disabling, initiating, and terminating load control; however, you can develop a SINAP/SS7 application in which individual application instances implement these functions. To do this, the application must be configured for load control individual-type operation. (For more information about load control individual-type operation, see the description of the `setup_req_t` structure's `type` field, which is described in `ca_setup_locon()` later in this chapter.)

NOTE

Load control cannot be implemented by an application that has one process performing both control and data processing, and another process with one or more instances also performing data processing at the same time. To implement load control, an application must have a separate control process, or it must have a process with several instances, one of which performs control processing.

Unless otherwise noted, the term *application* is used to refer to the application or application instance for which load control functionality is being implemented. The term *current application* refers to the application from which the load control function is being called. The term *current application instance* refers to the application instance from which the load control function is being called.

To implement load control functionality for an application, an application must call load control functions in the following order.

1. Call the `ca_register()` function to register with the SINAP/SS7 system.
2. Call the `ca_setup_locon()` function to configure the application for load control and to define load control operating characteristics.
3. Call the `ca_enable_locon()` function to initiate load control operation. This function causes the SINAP/SS7 system to begin monitoring the application for overload conditions.
4. Optionally, call the `ca_inquire_locon()` function to retrieve load control statistics.
5. Call the `ca_disable_locon()` function to terminate load control operation. the SINAP/SS7 system does not return to load control monitoring stage.

Using Load Control Keywords

Most load control functions contain the parameters `ssn` and `instance`, which specify the application and application instance, respectively, on which the function is to execute. For example, to indicate that a function call is to execute on the application whose SSN is `254`, you specify the value `254` for the function's *`ssn`* parameter.

NOTE

If the application uses enhanced message distribution, the application name is used instead of an SSN.

Many load control functions also allow you to define the value of the `ssn` or `instance` parameter by using a keyword, which represents a particular entity or group of entities. Each function description in this chapter indicates whether the `ssn` and `instance` parameters support the use of these keywords.

The following two keywords can be used to define the value of the `ssn` parameter.

- `SSN_THIS` specifies the current application. This value is typically used by an application that is implementing load control functionality for itself.
- `SSN_ALL` specifies all applications configured for load control. This value is typically used when a control program is being used to control load control operation across all SINAP/SS7 applications running on the module (system).

The following two keywords can be used to define the value of the `instance` parameter.

- `INST_THIS` specifies the current application instance. This value is typically used by an application instance that is implementing load control functionality for itself. To use this keyword, the application must be configured for load control individual-type operation. (For more information about individual-type operation, see the description of the `setup_req_t` structure's `type` field, which is described in `ca_setup_locon()` later in this chapter.)
- `INST_ALL` specifies all instances of the current application. To use this keyword, the application can be configured for either load control group- or individual-type operation. (For more information about load control group- and individual-type operation, see the description of the `setup_req_t` structure's `type` field, which is described in `ca_setup_locon()` later in this chapter.)

To execute a load control function on specific application instances, call the function once for each instance. For example, to disable load control for instances `1`, `3`, and `5` of the current application, call `ca_disable_locon()` three times, as shown in the following example.

```
ca_disable_locon(SSN_THIS, 1);
ca_disable_locon(SSN_THIS, 3);
ca_disable_locon(SSN_THIS, 5);
```

ca_disable_locon()

SYNOPSIS

```
int ca_disable_locon(int ssn, int instance);
```

INCLUDE FILE

```
$SINAP_HOME/Include/locon.h
```

DESCRIPTION

Load control is persistent and, once set up for an SSN or application, it remains set up even when the application is terminated or the SINAP/SS7 system is restarted. To remove the load control setting for an application, use the `ca_disable_locon()` function.

The `ca_disable_locon()` function terminates load control operation at the system, application, or instance level.

- Disabling load control at the system level terminates load control operation for all applications configured for load control. To disable load control at the system level, call `ca_disable_locon()` and specify the value `SSN_ALL` for the `ssn` parameter and `INST_ALL` for the `instance` parameter. The intention is that in a critical situation a control program can disable load control for all applications using a single function call.

  NOTE

  After disabling load control at the system level, you must re-enable load control at the same level before you can enable load control for an individual application. (You re-enable load control at the system level by calling `ca_enable_locon()` and specifying the value `SSN_ALL` for the `ssn` parameter and `INST_ALL` for the `instance` parameter.)

- Disabling load control at the application level terminates load control operation for all instances of a specific application. To disable load control at the application level, call `ca_disable_locon()`, specifying the application's SSN for the `ssn` parameter and `INST_ALL` for the `instance` parameter.
- Disabling load control at the instance level terminates load control operation for one or more instances of an application. To disable load control at the instance level, call

`ca_disable_locon()`, specifying the application's SSN for the `ssn` parameter and the number of the application instance for the `instance` parameter.

If the application uses enhanced message distribution, specify the name of the application instead of the SSN for the `SSN` parameter. To disable load control at the instance level, you must have specified the value `LC_INDIV` for the `setup_req_t` structure's `type` field. (For more information about this structure, see the description of `ca_setup_locon()` later in this chapter.)

NOTES

1. When an application stops running, load control cancels the disablement settings of individual application instances.
2. After disabling load control for specific application instances, you must re-enable load control for those same instances. Enabling load control at the application level does not override the enablement settings for individual application instances.

   For example, if you disable load control for instance `3` of the application whose SSN is `254`, you cannot re-enable load control for that instance by calling `ca_enable_locon()` and specifying the value `INST_ALL` for the `instance` parameter. To re-enable load control for instance `3`, you must call `ca_enable_locon()` and specify the value `3` for the `instance` parameter.
3. Load control for an instance is enabled only if all three levels are enabled. Therefore, disabling at any level disables load control for that instance.
4. Instance-level enablement is `ON` initially by default, and is always `ON` for group-type load control. The intention is to allow selective disablement for individual load control.

If you call this function while the SINAP/SS7 system is performing load control processing for an application, the SINAP/SS7 system extracts MSUs from the application's LIFO queue in FIFO fashion and appends them to the application's input queue. the SINAP/SS7 system discards MSUs that have been on the application's LIFO queue longer than the time defined in the `setup_req_t` structure's `delay` field (see the description of `ca_setup_locon()` later in this chapter for more information). The SINAP/SS7 system also discards any MSUs that would cause the application's input queue to overflow.

PARAMETERS

* `ssn`

  Specifies the SSN of the application for which you want to disable load control. Specify one of the following values.

  - Use a decimal number (in the range `2` through `255`) to specify a particular application by using an SSN. If the application is registered for enhanced message distribution, enter the 2- to 4-character application name instead of the SSN in the SSN field. The `ca_pack()` CASL function encodes the application name as a zero-filled, right-justified, U32 integer with a value greater than 255. If you specify `INST_ALL` for the `instance` parameter, the application need not be running.
  - Use the keyword `SSN_THIS` to specify the current application. If you specify `INST_ALL` for the `instance` parameter, you disable load control for all instances of the current application.
  - Use the keyword `SSN_ALL` to specify all applications that are configured for load control. You must specify `INST_ALL` for the `instance` parameter.

* `instance`

  Specifies the number of the application instance for which you want to disable load control. Specify one of the following values. If you specify `SSN_ALL` for the `ssn` parameter, you must specify `INST_ALL` for the `instance` parameter.

  - Use a decimal number (in the range `1` to `16`) to specify a particular application instance. You must have specified `LC_INDIV` for the `setup_req_t` structure's `type` field, and the instance must be running. (For more information about this structure, see the description of `ca_setup_locon()` later in this chapter.)
  - Use the keyword `INST_THIS` to specify the current application instance. You must have specified `LC_INDIV` for the `type` field of the `setup_req_t` structure. (For more information about this structure, see the description of `ca_setup_locon()` later in this chapter.)

    NOTE

    Use `INST_THIS` only if it is appropriate for this application instance to be implementing load control rather than the application's control process.

  - Use the keyword `INST_ALL` to specify all instances of the application.

RETURN VALUES

The `ca_disable_locon()` function returns the following values. The return value `-1` indicates an error. See Appendix C for information about load control errors.

VALUE	MEANING

`0`	Successful.
`-1`	Unsuccessful, `errno` indicates the error code. (The include file `$SINAP_HOME/Include/ca_error.h` contains CASL error codes and messages; `/sys/errno.h` contains UNIX error codes and messages.)

ca_enable_locon()

SYNOPSIS

```
int ca_enable_locon(int ssn, int instance);
```

INCLUDE FILE

```
$SINAP_HOME/Include/locon.h
```

DESCRIPTION

The `ca_enable_locon()` function initiates load control operation for the specified application. Calling this function causes the SINAP/SS7 system to begin monitoring the specified application for overload conditions. During this monitoring stage, load control is active; however, the SINAP/SS7 system does not actively perform load control processing until overload conditions occur.

For the SINAP/SS7 system to perform load control processing, load control must be enabled at the system, application, and instance levels (see the following list). By default, the `ca_enable_locon()` function automatically enables load control at the system and instance levels. You call `ca_enable_locon()` to complete the enablement process and enable load control at the application level (i.e., you enable load control for a specific application). However, if you disable load control at the system or instance level, you cannot complete the enablement process by simply enabling load control at the application level; instead, you must first re-enable load control at the level for which it was disabled (system or instance). (For more information, see the description of `ca_disable_locon()` earlier in this chapter.)

- Enabling load control at the system level initiates load control operation for all applications configured for load control. To enable load control at the system level, call `ca_enable_locon()` and specify the value `SSN_ALL` for the `ssn` parameter and `INST_ALL` for the `instance` parameter.
- Enabling load control at the application level initiates load control operation for all instances of a specific application. To enable load control at the application level, call `ca_enable_locon()`, specifying the application's SSN for the `ssn` parameter and `INST_ALL` for the `instance` parameter. If the application is registered for enhanced message distribution, enter the 2- to 4-character application name instead of the SSN in the SSN field. The `ca_pack()` CASL function encodes the application name as a zero-filled, right-justified, U32 integer with a value greater than 255.

- Enabling load control at the instance level initiates load control operation for one or more instances of a specific application. To enable load control at the instance level, call `ca_enable_locon()`, specifying the application's SSN for the `ssn` parameter and the number of the application instance for the `instance` parameter. To use this form of the `ca_enable_locon()` function, you must have specified the value `LC_INDIV` for the `setup_req_t` structure's `type` field. (For more information about this structure, see the description of `ca_setup_locon()` later in this chapter.)

  NOTE

  Load control for an instance is enabled only if all three levels are enabled.

You cannot configure individual application instances for load control. However, you can enable load control for a **subset** of the application's instances, thereby selectively implementing load control processing for specific application instances. For example, suppose you configure an application for load control and then enable load control for instances `1`, `3`, and `5` only. If all of the application's instances begin experiencing overload conditions, the SINAP/SS7 system is able to perform load control processing for instance `1`, `3`, and `5` only.

PARAMETERS

* `ssn`

Specifies the SSN of the application for which you want to enable load control. The application must be configured for load control. If you specify `INST_ALL` for the `instance` parameter, the application need not be running. Specify one of the following values.

- Use a decimal number (in the range `2` to `255`) to specify a particular application using an SSN value.
- If the application is registered for enhanced message distribution, enter the 2- to 4-character application name instead of the SSN in the SSN field. The `ca_pack()` CASL function encodes the application name as a zero-filled, right-justified, U32 integer with a value greater than 255.
- Use the keyword `SSN_THIS` to specify the current application.
- Use the keyword `SSN_ALL` to specify all applications configured for load control. You must specify `INST_ALL` for the `instance` parameter.

* `instance`

Specifies the instance number of the application instance for which you want to enable load control. Specify one of the following values. If you specify `SSN_ALL` for the `ssn` parameter, you must also specify `INST_ALL` for the `instance` parameter.

- Use a decimal number (in the range `1` to `16`) to specify a particular application instance. You must have specified the value `LC_INDIV` for the `setup_req_t`

structure's `type` field, and the instance must be running. (For more information about this structure, see the description of `ca_setup_locon()` later in this chapter.)

- Use the keyword `INST_THIS` to specify the current application instance. You must have specified the value `LC_INDIV` for the `setup_req_t` structure's `type` field. (For more information about this structure, see the description of `ca_setup_locon()` later in this chapter.)

  NOTE

  Use the keyword `INST_THIS` only if it is appropriate for this application instance to be implementing load control rather than the application's control process.

- Use the keyword `INST_ALL` to specify all instances of the application.

RETURN VALUES

The `ca_enable_locon()` function returns the following values. The return value `-1` indicates an error. See Appendix C for information about load control errors.

VALUE	MEANING
`0`	Successful.
`-1`	Unsuccessful, `errno` indicates the error code. (The include file `$SINAP_HOME/Include/ca_error.h` contains CASL error codes and messages; `/sys/errno.h` contains UNIX error codes and messages.)

ca_exit_locon()

SYNOPSIS

```
int ca_exit_locon(int ssn, int instance);
```

INCLUDE FILE

```
$SINAP_HOME/Include/locon.h
```

DESCRIPTION

The `ca_exit_locon()` function deactivates load control processing for the specified application. You can call this function only if you called the `ca_invoke_locon()` function to invoke load control processing for the application; otherwise, an error occurs.

After this function executes, load control is still enabled; therefore, the SINAP/SS7 system returns to monitoring the application for overload conditions. To completely terminate load control operation, you must call the `ca_disable_locon()` function.

Calling the `ca_exit_locon()` function while the SINAP/SS7 system is performing load control processing for the application causes the SINAP/SS7 system to extract MSUs from the application's LIFO queue in FIFO fashion and append them to the application's input queue. The SINAP/SS7 system discards MSUs that have been on the application's LIFO queue longer than the time defined in the `setup_req_t` structure's `abate_delay` field (see the description of `ca_setup_locon()` later in this chapter). The SINAP/SS7 system also discards any MSUs that would cause the application's input queue to overflow.

NOTE

When you call `ca_exit_locon()` to deactivate load control processing, the SINAP/SS7 system does not check for overload conditions until it processes the next incoming MSU. If this functionality is unacceptable for your application, then do not call `ca_invoke_locon()` to activate load control processing for the application.

PARAMETERS

* `ssn`

  Specifies the SSN of the application for which you want to deactivate load control processing. The application must be running. Specify one of the following values.

  - Use a decimal number (in the range `2` through `255`) to specify a particular application using an SSN.
  - If the application is registered for enhanced message distribution, enter the 2- to 4-character application name instead of the SSN in the SSN field. The `ca_pack()` CASL function encodes the application name as a zero-filled, right-justified, U32 integer with a value greater than 255.
  - Use the keyword `SSN_THIS` to specify the current application.

* `instance`

  Specifies the number of the application instance for which you want to deactivate load control processing. Specify one of the following values.

  - Use a decimal number (in the range `1` through `16`) to specify a particular application instance. You must have specified the value `LC_INDIV` for the `setup_req_t` structure's `type` field, and the instance must be running. (For more information about this structure, see the description of `ca_setup_locon()` later in this chapter.)
  - Use the keyword `INST_THIS` to specify the current application instance. You must have specified the value `LC_INDIV` for the `setup_req_t` structure's `type` field. (For more information about this structure, see the description of `ca_setup_locon()` later in this chapter.)

    NOTE

    Use the keyword `INST_THIS` only if it is appropriate for this application instance to be implementing load control rather than the application's control process.

  - Use the keyword `INST_ALL` to specify all instances of the application. You can use this keyword whether you specified the value `LC_INDIV` or `LC_GROUP` for the `setup_req_t` structure's `type` field. (For more information about this structure, see the description of `ca_setup_locon()` later in this chapter.)

RETURN VALUES

The `ca_exit_locon()` function returns the following values. The return value `-1` indicates an error. See Appendix C for information about load control errors.

VALUE	MEANING
`0`	Successful.
`-1`	Unsuccessful, `errno` indicates the error code. (The include file `$SINAP_HOME/Include/ca_error.h` contains CASL error codes and messages; `/sys/errno.h` contains UNIX error codes and messages.)

ca_inquire_locon()

SYNOPSIS

```
int ca_inquire_locon(inquire_req_t *p_inquire);
```

INCLUDE FILE

```
$SINAP_HOME/Include/locon.h
```

DESCRIPTION

The `ca_inquire_locon()` function retrieves load control statistics for the specified application. The `ca_inquire_locon()` function returns information about the specified application in the fields of the `inquire_req_t` structure. If the application is configured for load control individual-type operation (that is, you specified the value `LC_INDIV` for the `setup_req_t` structure's `type` field), `ca_inquire_locon()` returns information about application instances in the `instance_state_t` structure.

Before calling `ca_inquire_locon()`, initialize the `ssn` field of the `inquire_req_t` structure to the SSN of the application whose load control statistics you want to retrieve.

PARAMETERS

* `p_inquire` (input)
  Specifies a pointer to a data structure of the following type.

```
typedef struct inquire_req_s
{
   S32       ssn;
   U8        type;
   U8        notify;
   S16       threshold;
   S16       delay;
   S16       count;
   S16       abate_delay;
   U8        enabled;
   U8        state;
   U8        system_enabled;
   U8        instance_count;
   instance_state_t   instance[MAX_INSTANCES];
```

```
} inquire_req_t;
```

* `ssn` (input)
  Specifies the SSN of the application whose load control statistics you want to retrieve.

  - Initialize this field to a decimal number (in the range `2` through `255`) to specify a particular application using an SSN value. The application need not be running.
  - If the application is registered for enhanced message distribution, enter the 2- to 4-character application name instead of the SSN in the SSN field. The `ca_pack()` CASL function encodes the application name as a zero-filled, right-justified, U32 integer with a value greater than 255.
  - Initialize this field to the value `SSN_THIS` to specify the current application. The `ca_inquire_locon()` function replaces `SSN_THIS` with the actual SSN of the current application.

  If the call is successful, `ca_inquire_locon()` fills in the fields of the `inquire_req_t` structure, as follows.

* `type` (output)
  Indicates the type of load control operation currently in effect. (For detailed explanations of these values, see the description of the `setup_req_t` structure's `type` field, which is described in `ca_setup_locon()` later in this chapter.)

  - `LC_GROUP` indicates that the application is configured for load control group-type operation, which means that load control treats all instances of the application as a single entity.
  - `LC_INDIV` indicates that the application is configured for load control individual-type operation, which means that load control treats each application instance as a separate entity.
  - `LC_DELETE` indicates that load control functionality has been removed from the application.

* `notify` (output)
  Indicates whether the application is to be notified when load control is initiated and terminated and load control processing is activated and deactivated. (For detailed explanations of these values, see the description of the `setup_req_t` structure's `notify` field, which is described in `ca_setup_locon()` later in this chapter.)

  - `LC_NOTIFY` indicates that the application is configured to receive notification when one of these actions occurs.
  - `LC_NONOTIFY` indicates that the application is not configured to receive notification when one of these actions occurs.

  The following fields indicate the values of the `setup_req_t` structure's `threshold`, `delay`, `count`, and `abate_delay` fields. For more information about this structure, see `ca_setup_locon()` later in this chapter.

* `threshold` (output)
  Indicates the maximum number of MSUs allowed on the application's input queue at any one time.

* `delay` (output)
  Indicates the maximum amount of time (in milliseconds) within which the application must process an incoming MSU. To disable the MSU delay count and not use it as a consideration for determining load control onset, set `delay` to zero. If `delay` equals zero, `count` must also be set to zero, or an error occurs. The `abate_delay` parameter must be set to a positive value.

* `count` (output)
  Indicates the maximum number of consecutive outbound MSUs that the application is allowed before the SINAP/SS7 system activates load control processing.

  To disable the MSU delay count and not use it as a consideration for determining load control onset, set `count` to zero. If `count` is set to zero, `delay` must also be set to zero or an error occurs. The `abate_delay` parameter must be set to a positive value.

* `abate_delay` (output)
  Indicates the maximum amount of time (in milliseconds) that an MSU can spend on the application's LIFO queue during load control processing.

* `enabled` (output)
  Indicates whether load control is enabled for the application.

  - `LC_ENABLED` indicates that load control is enabled.
  - `LC_DISABLED` indicates that load control is disabled.

* `state` (output)
  Indicates the application's current load control status.

  - `LC_NOTRUN` indicates that the application is not currently running.
  - `LC_RUN` indicates that the application is currently running and that the SINAP/SS7 system is not currently performing load control processing for the application.
  - `LC_AUTO` indicates that the application is running and has called `ca_enable_locon()`, which allows the SINAP/SS7 system to activate load control processing automatically when the application experiences overload conditions. (This value applies only if load control is configured for group-type operation. See the description of the `type` field.)
  - `LC_FORCE` indicates that the application is running and has called `ca_invoke_locon()`, which causes the SINAP/SS7 system to begin performing load control processing, even if the application is not experiencing overload conditions. (This value applies only if load control is configured for group-type operation. See the description of the `type` field.)

* `system_enabled` (output)
  Indicates whether load control is enabled at the system level. (For information about the levels at which load control can be enabled, see `ca_enable_locon()` earlier in this chapter.)

  - `LC_ENABLED` indicates that load control is enabled at the system level.
  - `LC_DISABLED` indicates that load control is disabled at the system level.

* `instance_count` (output)
  Indicates the number of application instances currently running. If the `type` field of the `inquire_req_t` structure is set to `LC_INDIV` (which indicates that the application is configured for load control individual-type operation), `ca_inquire_locon()` returns the following additional information.

* `instance[MAX_INSTANCES]` (output)
  An array of `instance_state_t` structures that the SINAP/SS7 system obtains, where the index
  `x < MAX_INSTANCES (16)` and `instance number = x + 1`.
  (`MAX_INSTANCES` is defined in the `sinap.h` include file.)

```
typedef struct instance_state_s
{
   U8        enabled;
   U8        state;
   U16       filler;
   pid_t     pid;
} instance_state_t;
```

* `enabled` (output)
  Indicates whether load control is enabled at the instance level. (For information about the levels at which load control can be enabled, see `ca_enable_locon()` earlier in this chapter.)

  - `LC_ENABLED` indicates that load control is enabled for this application instance.
  - `LC_DISABLED` indicates that load control is disabled for this application instance.

* `state` (output)
  Indicates the application instance's current load control status.

  - `LC_NOTRUN` indicates that the application instance is not currently running.
  - `LC_RUN` indicates that the application instance is currently running and that the SINAP/SS7 system is not currently performing load control processing for the application instance.
  - `LC_AUTO` indicates that the application instance is running and has called `ca_enable_locon()`, which allows the SINAP/SS7 system to activate load

control processing automatically when the application instance experiences overload conditions.

- `LC_FORCE` indicates that the application instance is running and has called `ca_invoke_locon()`, which causes the SINAP/SS7 system to begin performing load control processing, even if the application is not experiencing overload conditions.

* `filler` (output)
  This field is used internally by the SINAP/SS7 system. Do not modify it.

* `pid` (output)
  Indicates the process ID (PID) that the SINAP/SS7 system assigned to the application instance.

RETURN VALUES

The `ca_inquire_locon()` function returns the following values. The return value `-1` indicates an error. See Appendix C, ‘‘CASL Error Messages,” for information about load control errors.

VALUE	MEANING
`0`	Successful.
`-1`	Unsuccessful, `errno` indicates the error code. (The include file `$SINAP_HOME/Include/ca_error.h` contains CASL error codes and messages; `/sys/errno.h` contains UNIX error codes and messages.)

ca_invoke_locon()

SYNOPSIS

```
int ca_invoke_locon(int ssn, int instance);
```

INCLUDE FILE

```
$SINAP_HOME/Include/locon.h
```

DESCRIPTION

The `ca_invoke_locon()` function causes the SINAP/SS7 system to begin performing load control processing for the specified application, even if the application is **not** experiencing overload conditions. The SINAP/SS7 system continues to perform load control processing for the application until you call the `ca_disable_locon()` or `ca_exit_locon()` function, issue the `DISABLE-LOAD-CONTROL` or `EXIT-LOAD-CONTROL` command, or terminate the application. (See Appendix A, ‘‘SINAP/SS7 MML Command Summary,” of this manual or see the *SINAP/SS7 User’s Guide* (R8051) for information about these MML commands.)

Stratus does not recommend using the `ca_invoke_locon()` function for normal operation. Instead, use the `ca_enable_locon()` function, which allows the SINAP/SS7 system to activate load control processing automatically when the application experiences overload conditions.

PARAMETERS

* `ssn` (input)

  Specifies the SSN of the application for which you want the SINAP/SS7 system to begin performing load control processing. The application **must** be running. Specify one of the following values.

  - Use a decimal number (in the range `2` through `255`) to specify a particular application by using an SSN value.
  - If the application is registered for enhanced message distribution, enter the 2- to 4-character application name instead of the SSN in the SSN field. The `ca_pack()` CASL function encodes the application name as a zero-filled, right-justified, U32 integer with a value greater than 255.
  - Use the keyword `SSN_THIS` to specify the current application.

* `instance` (input)

  Specifies the number of the application instance for which you want the SINAP/SS7 system to begin performing load control processing. Specify one of the following values.

  - Use a decimal number (in the range `1` through `16`) to specify a particular application instance. You must have specified the value `LC_INDIV` for the `setup_req_t` structure's `type` field, and the instance must be running. (For more information about this structure, see the description of `ca_setup_locon()` later in this chapter.)
  - Use the keyword `INST_THIS` to specify the current application instance. You must have specified the value `LC_INDIV` for the `setup_req_t` structure's `type` field. (For more information about this structure, see the description of `ca_setup_locon()` later in this chapter.)

    NOTE

    Use the keyword `INST_THIS` only if it is appropriate for this application instance to be implementing load control rather than the application's control process.

  - Use the keyword `INST_ALL` to specify all instances of the application. You can use this keyword whether you specified the value `LC_INDIV` or `LC_GROUP` for the `setup_req_t` structure's `type` field. (For more information about this structure, see the description of `ca_setup_locon()` later in this chapter.)

RETURN VALUES

The `ca_invoke_locon()` function returns the following values. The return value `-1` indicates an error. See Appendix C for information about load control errors.

VALUE	MEANING
`0`	Successful.
`-1`	Unsuccessful, `errno` indicates the error code. (The include file `$SINAP_HOME/Include/ca_error.h` contains CASL error codes and messages; `/sys/errno.h` contains UNIX error codes and messages.)

ca_setup_locon()

SYNOPSIS

```
int ca_setup_locon(setup_req_t *p_setup);
```

INCLUDE FILE

```
$SINAP_HOME/Include/locon.h
```

DESCRIPTION

The `ca_setup_locon()` function configures an application for load control and defines load control operating characteristics. You must issue a separate `ca_setup_locon()` function call for each application that you want to configure for load control. After calling `ca_setup_locon()`, you must either call `ca_enable_locon()` or issue the MML command `ENABLE-LOAD-CONTROL` to initiate load control operation for the application. (See Appendix A of this manual or see the *SINAP/SS7 User's Guide* (R8051) for information about the MML command `ENABLE-LOAD-CONTROL`.)

To define load control operating characteristics for the application, you must initialize the fields in the `setup_req_t` structure before calling `ca_setup_locon()`. To modify an application's existing load control operating characteristics, call `ca_setup_locon()` with the fields of the `setup_req_t` structure initialized to the desired new values. To remove load control functionality from an application, call `ca_setup_locon()` with the `type` field of `setup_req_t` initialized to the value `LC_DELETE`.

The application's load control operating characteristics are stored in a static database and they remain in effect until you change them, or until you reinitialize the system. The fields `threshold`, `delay`, and `count` define the application's acceptable level of network congestion. The default criteria for determining load control onset is to evaluate both the MSU delay count and the length of the input queue versus the threshold. To disable the MSU delay count and use the input queue length as the sole determining factor, set both `count` and `delay` to zero and set `abate_delay` to a positive value.

NOTE

If the application used enhanced message distribution, use the 2- to 4-character application name instead of the SSN in the SSN field. The `ca_pack()` CASL function encodes the

application name as a zero-filled, right-justified, U32 integer with a value greater than 255.

The LIFO queue that the SINAP/SS7 system creates for load control processing is the same size as the application's input queue, which is defined by the `register_req_t` structure's `max_msu_input_que` field.

The application's current state has the following effects on the load control operating characteristics defined by the `ca_setup_locon()` function call.

- If you are configuring an application for load control and the application is active when you call this function, load control operating characteristics take effect when the function call has finished executing. The SINAP/SS7 system implements load control for the specified application without disrupting the application's normal processing.
- If you are configuring the application for load control individual-type operation, load control operating characteristics take effect when an application instance calls the `ca_register()` function to register with the SINAP/SS7 system.
- If you are calling this function to modify load control characteristics for an application, the new characteristics take effect when the function call has finished executing. If the SINAP/SS7 system is currently performing load control processing for the application, the SINAP/SS7 system validates the new characteristics without disrupting active load control processing.
- If you remove load control from an application for which the SINAP/SS7 system is currently performing load control processing, the SINAP/SS7 system extracts MSUs from the application's LIFO queue in FIFO fashion and appends them to the application's input queue. The SINAP/SS7 system discards MSUs that have been on the application's LIFO queue longer than the time defined by the `ca_setup_locon()` function's `abate_delay` parameter. The SINAP/SS7 system also discards any MSUs that would cause the application's input queue to overflow.

PARAMETERS

* `p_setup` (input)

Specifies a pointer to a data structure of the following type. Before calling `ca_setup_locon()`, initialize this structure's fields to appropriate values.

NOTE

Specifying values that are too low can cause unpredictable load control behavior.

The **`setup_req_t`** structure is as follows:

```
typedef struct setup_req_s
{
   S32     ssn;
   U8      type;
   U8      notify;
   S16     threshold;
   S16     delay;
   S16     count;
   S16     abate_delay;
} setup_req_t;
```

* `ssn` (input)
  Specifies the SSN of the application being configured for load control.

  - Use a decimal number (in the range `2` through `255`) to specify a particular application using the SSN. The application need not be running.
  - If the application is registered for enhanced message distribution, enter the 2- to 4-character application name instead of the SSN in the SSN field. The `ca_pack()` CASL function encodes the application name as a zero-filled, right-justified, U32 integer with a value greater than 255.
  - Use the keyword `SSN_THIS` to specify the current application. The `ca_setup_locon()` function replaces `SSN_THIS` with the actual SSN of the current application.

* `type` (input)
  Specifies how the SINAP/SS7 system evaluates the network congestion levels of individual application instances to determine if the application is experiencing overload conditions. Initialize this field to one of the following values.

  NOTE

  Stratus recommends specifying `LC_GROUP`, which causes the SINAP/SS7 system to evaluate the application's network congestion based on the combined congestion levels of each of the application's instances. If you plan to allow individual application instances to enable, disable, invoke, and exit load control, you **must** initialize this field to `LC_INDIV`.

  - `LC_GROUP` (load control group-type operation) specifies that the SINAP/SS7 system is to treat all of the application's instances as a single entity. The SINAP/SS7 system activates load control processing only when **all** of the application's instances are experiencing overload conditions. For example, if an application has 10 instances, and only 3 of the 10 are experiencing overload conditions, the SINAP/SS7 system does not

activate load control processing for any of the instances; all 10 instances must be experiencing overload conditions.

You can use the value `LC_GROUP` whether the `inbound_load_dist_type` field of the `register_req_t` structure is set to a value of `1` or `2`. (The value `1` specifies round-robin load distribution; the value `2` specifies least-utilized load distribution. For more information about this structure, see the description of `ca_register()` earlier in this chapter.)

- `LC_INDIV` (load control individual-type operation) specifies that the SINAP/SS7 system is to treat each of the application's instances as a separate entity. The SINAP/SS7 system activates load control processing for an application instance whenever that instance experiences overload conditions.

  To use `LC_INDIV`, the `inbound_load_dist_type` field of the `register_req_t` structure **must** be set to a value of `1`, which specifies round-robin load distribution. (For more information about this structure, see the description of `ca_register()` earlier in this chapter.)

- `LC_DELETE` removes load control functionality from the application. You cannot remove load control functionality from individual application instances.

* `notify` (input)
Specifies whether the SINAP/SS7 system notifies the application when load control is initiated and terminated and when load control processing is activated and deactivated. Initialize this field to one of the following values.

- `LC_NOTIFY` specifies that the application should be notified. The SINAP/SS7 system uses an IPC message to notify the application. (For more information about IPC messages, see the section "Interprocess Communications (IPC)" in Chapter 2.) The IPC message type is `I_NOTIFICATION` and its data format is shown below. (Note that the data format is defined in `$SINAP_HOME/Include/locon.h`.)

```
typedef struct lc_notify_s
{
      U8     ssn;
      U8     instance;      /* 0 for type LC_GROUP */
      U8     state;         /* LC_AUTO/LC_FORCE */
      U8     action;        /* LC_INIT/LC_ABATE */
} lc_notify_t
```

- `LC_NONOTIFY` specifies that the application should not be notified.

* `threshold` (input)
Specifies the maximum number of MSUs allowed on the application's input queue. Initialize this field to a decimal number in the range `1` to the value of the `register_req_t` structure's `max_msu_input_que` field (not to exceed `10000`). (For more information about this structure, see the description of `ca_register()` earlier in this chapter.)

The SINAP/SS7 system activates load control processing when the number of MSUs on the input queue meet or exceed this value, **and** the values of the *delay* and *count* fields are met or exceeded.

* *delay* (input)
  Specifies the amount of time (in milliseconds) within which the application must process an incoming MSU. Initialize this field to a decimal number in the range 1 through 10000.

  The SINAP/SS7 system activates load control processing when the number of incoming MSUs defined by the *count* field have not been processed within this amount of time, **and** the number of MSUs on the application's input queue meet or exceed the value defined in the *threshold* field. Note that MSU processing time is measured from the time an incoming MSU arrives on the input queue to the time the application places a response on the output queue.

  NOTE

  To make use of the timestamp applied to incoming MSUs (which the SINAP/SS7 system uses to monitor an application for overload conditions), an application **must** use the same t_block_t structure for the response as was used by the incoming MSU; otherwise, the timestamp is rendered useless.

  To disable the MSU delay count and not use it as a criteria for determining load control onset, set delay to zero. The count parameter must also be set to zero or an error occurs, specifying the nonzero field. The abate_delay parameter must be set to a positive value.

* count (input)
  Specifies the maximum number of consecutive outbound MSUs that the application is allowed. Initialize this field to a decimal number in the range 1 through 10000.

  The SINAP/SS7 system activates load control processing when the number of MSUs on the application's output queue meet or exceed this value, **and** the number of incoming MSUs on the application's input queue meet or exceed the value defined in the *threshold* field.

  To disable the MSU delay count and not use it as a criteria for determining load control onset, set count to zero. The delay parameter must also be set to zero or an error occurs, specifying the nonzero field. The abate_delay parameter must be set to a positive value.

* abate_delay (input)
  Specifies the maximum amount of time (in milliseconds) that an MSU can spend on the application's LIFO queue during load control processing. Initialize this field to a decimal number in the range 1 through 10000.

When the SINAP/SS7 system extracts an MSU from the LIFO queue, it compares the length of time that the MSU has been on the queue to the value defined by this field. If the MSU's time on the queue meets or exceeds this value, the SINAP/SS7 system discards all of the MSUs on the LIFO queue, since they have been on the queue too long.

RETURN VALUES

The `ca_setup_locon()` function returns the following values. The return value `-1` indicates an error. See Appendix C for information about load control errors.

VALUE	MEANING
`0`	Successful.
`-1`	Unsuccessful, `errno` indicates the error code. (The include file `$SINAP_HOME/Include/ca_error.h` contains CASL error codes and messages; `/sys/errno.h` contains UNIX error codes and messages.)

BITE Functions

This section contains an alphabetic reference of the following CASL functions, which can be used in any type of application in order to implement testing options.

- `ca_dbg_display()`
- `ca_dbg_dump()`
- `ca_disable_intc()`
- `ca_disable_mon()`
- `ca_enable_intc()`
- `ca_enable_mon()`

ca_dbg_display()

SYNOPSIS

```
int     ca_dbg_display(
          char     *pstring);
```

DESCRIPTION

The `ca_dbg_display()` function sends debug messages as ASCII strings to the BITE for logging. If the Terminal Handler is currently monitoring the process, `ca_dbg_display()` also sends the messages to the terminal to aid in problem solving.

If monitoring is enabled, the `ca_dbg_display()` function sends debug messages as ASCII strings to the process's log file. If monitoring is not enabled, the process sends the ASCII strings to the BITE's default log file. If a string is longer than 255 bytes, a null is inserted after the 255th byte and the remaining bytes are discarded.

The debug message does not cause an event that is visible to Node Management, and is not recorded in the Trouble Management log or in the Stratus console/system log.

PARAMETERS

* `pstring` (input)
  Specifies a pointer to the null-terminated ASCII string being logged. The string should not exceed 255 bytes.

FILES

`arch.h`, `ca_error.h`

RETURN VALUES

The `ca_dbg_display()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

Possible CASL values for `errno` are as follows.

Value	Meaning
`CA_ERR_ACCESS`	The process is not registered. Call `ca_register()` before calling this function.
`CA_ERR_DESTN_KEY`	Destination process key not found.
`CA_ERR_MSG_TRUNCATED`	String is truncated to 255 bytes.

This function performs a `ca_put_msg()` and can also return the errors listed under that function.

SEE ALSO

`ca_dbg_dump()`

ca_dbg_dump()

SYNOPSIS

```
int     ca_dbg_dump(
          U8      *pdump,
          U16     size);
```

DESCRIPTION

The `ca_dbg_dump()` function sends the specified portion of memory to the BITE for log file recording. If the Terminal Handler is currently monitoring the process, messages also appear at the terminal.

If monitoring is enabled, the function sends the data as ASCII strings to the process's log file. If monitoring is not enabled, the process sends the ASCII strings to the BITE's default log file.

PARAMETERS

* `pdump` (input)
  Specifies a pointer to the requested area of memory.

* `size` (input)
  Specifies the size of the requested area of memory.

FILES

`arch.h`, `ca_error.h`

RETURN VALUES

The `ca_dbg_dump()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

Possible CASL values for `errno` are as follows.

Value	Meaning
`CA_ERR_ACCESS`	The process is not registered. Call `ca_register()` before calling this function.
`CA_ERR_DESTN_KEY`	Destination process key not found.

This function performs a `ca_put_msg()` and can also return the errors listed under that function.

SEE ALSO

`ca_dbg_display()`

ca_disable_intc()

SYNOPSIS

```
int     ca_disable_intc();
```

DESCRIPTION

The `ca_disable_intc()` function instructs the BITE to discontinue scenario execution (known as *intercept mode*) and return the calling process to normal network communications activity. To stop scenario execution, a process must call this function. The function does not have any parameters.

FILES

`arch.h`, `ca_error.h`

RETURN VALUES

The `ca_disable_intc()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

Possible CASL values for `errno` are as follows.

Value	Meaning
`CA_ERR_ACCESS`	The process calling `ca_disable_intc()` is not registered. Call `ca_register()` before calling this function
`CA_ERR_DESTN_KEY`	Destination process key not found.

This function calls `ca_get_msg()` and `ca_put_msg()` and can return the errors listed under those functions.

SEE ALSO

`ca_enable_intc()`

ca_disable_mon()

SYNOPSIS

```
int     ca_disable_mon();
```

DESCRIPTION

The `ca_disable_mon()` function instructs the BITE to discontinue monitoring activity for the calling process. The function does not have any parameters.

FILES

`arch.h`, `ca_error.h`

RETURN VALUES

The `ca_disable_mon()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

Possible CASL values for `errno` are as follows.

Value	Meaning
`CA_ERR_ACCESS`	The process calling `ca_disable_mon()` is not registered. Call `ca_register()` before calling this function
`CA_ERR_DESTN_KEY`	Destination process key not found.

This function performs a `ca_put_msg()`and a `ca_get_msg()` and can also return the errors listed under those functions.

SEE ALSO

`ca_enable_mon()`

ca_enable_intc()

SYNOPSIS

```
int     ca_enable_intc(
          char    *ps);
```

DESCRIPTION

The `ca_enable_intc()` function enables scenario execution (intercept mode). The function instructs the BITE to place the SS7 communication path to the calling process in intercept mode, and then simulates network activity by starting a scenario execution program under the BITE's control.

The calling process provides the fully qualified file name of the desired scenario execution program. If the calling process had previously enabled monitoring, the result of the scenario execution is logged.

NOTE

You can also enable scenario execution by specifying intercept mode in the `ca_register()` function or by issuing the MML command `START-SCEN`.

PARAMETERS

* `ps` (input)
  Specifies a pointer to the path name of the desired scenario execution program.

FILES

`arch.h`, `ca_error.h`

RETURN VALUES

The `ca_enable_intc()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

Possible CASL values for `errno` are as follows.

Value	Meaning
`CA_ERR_ACCESS`	The process calling `ca_enable_intc()` is not registered. Call `ca_register()` before calling this function
`CA_ERR_DESTN_KEY`	Destination process key not found.
`CA_ERR_INT_MML`	Error in command to BITE.

This function calls `ca_get_msg()` and `ca_put_msg()` and can return the errors listed under those functions.

SEE ALSO

`ca_disable_intc()`

ca_enable_mon()

SYNOPSIS

```
int   ca_enable_mon(
        BOOL     ipc,
        BOOL     ss7,
         U8       *pfn);
```

DESCRIPTION

To enable monitoring, a process calls the `ca_enable_mon()` function. This function activates BITE monitoring of IPC or SS7 activity.

You can also enable monitoring by specifying `fmon_ipc` and/or `fmon_ss7` in the `ca_register()` function or by issuing the MML command `START-MON`.

PARAMETERS

* `ipc` (input)
  Specifies whether IPC activities are to be monitored. Specify a `1` to monitor IPC activities; otherwise, specify `0`.

* `ss7` (input)
  Specifies whether SS7 activities are to be monitored. Specify a `1` to monitor SS7 activities; otherwise, specify `0`.

* `pfn` (input)
  Specifies a pointer to the path name of the log file to which monitoring results are to be written.

FILES

`arch.h`, `ca_error.h`

RETURN VALUES

The `ca_enable_mon()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

Possible CASL values for `errno` are as follows.

Value	Meaning
`CA_ERR_ACCESS`	The process is not registered. Call `ca_register()` before calling this function
`CA_ERR_DESTN_KEY`	Destination process key not found.
`CA_ERR_INT_MML`	Error in command to BITE.

This function calls `ca_get_msg()` and `ca_put_msg()` and can return the errors listed under those functions.

SEE ALSO

`ca_disable_mon()`

Miscellaneous Functions

This section contains an alphabetic reference of the following CASL functions, which can be used in any type of application.

- `ca_health_chk_req()`
- `ca_health_chk_resp()`
- `ca_pack()`
- `ca_put_event()`
- `ca_unpack()`

ca_health_chk_req()

SYNOPSIS

```
int    ca_health_chk_req(
         ipc_key_t      destn_key);
```

DESCRIPTION

The `ca_health_chk_req()` function sends a health-check request to the specified destination, using the specified IPC key. The destination must respond to this request within the amount of time specified by the SINAP/SS7 environment variable `SINAP_HEALTH_TIMEOUT`. Otherwise, the health-check request fails, prompting trouble management to perform the procedure defined in the trouble treatment table (see the *SINAP/SS7 User's Guide* (R8051) for more information).

To receive health-check requests, the destination must be registered to receive them (i.e., it must have called the `ca_register()` function with the `fhealth_check_option` parameter set to `1`).

NOTE

Application processes can use health-check messages to poll one another.

PARAMETERS

* `destn_key` (input)
  Specifies the `ipc_key_t` structure that contains the IPC key of the destination process. To use `ca_health_chk_req()`, you must assign values to the fields in the `ipc_key_t` structure, which is described in the following section "The IPC Key Structure (`ipc_key_t`)."

NOTE

The SINAP/SS7 system returns an error if the specified IPC key is invalid.

IPC Key Structure (`ipc_key_t`)

The `ipc_key_t` structure contains the following fields and is defined in the include file `sinap.h`.

```
typedef struct ipc_key_s
{
          U32      node;
          U32      module;
          U32      appl;
          U32      proc;
          U8       instance;
          U8       node_index;
          U16      ipc_index;
} ipc_key_t;
```

* `node` (output)
  Specifies the ID of the SINAP node on which your application is running. You can determine this value from the `NODE=` entry in the `/etc/sinap_master` file. You should modify any script files or user-defined program files that contain an invalid, hard-coded node name.

* `module` (output)
  Specifies the name or ID of the module. You can determine this value from the `MODULE=` entry in the `/etc/sinap_master` file. You should modify any script files or user-defined program files that contain an invalid, hard-coded module name.

* `appl` (input)
  Specifies the compressed application ID.

* `proc` (input)
  Specifies the compressed process ID.

* `instance` (input)
  Specifies the instance ID (in the range `1` through `8`). A value of `0` indicates that the field is not used.

* `node_index` (input)
  Specifies the index (0 through 3) of the node.

* `ipc_index` (input)
  Specifies the index ID of the IPC process table.

FILES

`arch.h`, `ca_error.h`, `sinap.h`

RETURN VALUES

The `ca_health_chk_req()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

A possible CASL value for `errno` is as follows.

Value	Meaning
`CA_ERR_ACCESS`	The process calling `ca_health_chk_req()` is not registered. The process must call `ca_register()` before calling this function.

This function performs a `ca_put_msg()` and can also return the errors listed under that function.

SEE ALSO

`ca_put_msg()`, `ca_health_chk_resp()`, `ca_register()`

ca_health_chk_resp()

SYNOPSIS

```
int     ca_health_chk_resp(
          i_block_t      *piblk);
```

DESCRIPTION

The `ca_health_chk_resp()` function is used to respond to a health-check request. When a process receives a health-check request, it must call this function within the amount of time specified in the SINAP/SS7 environment variable `SINAP_HEALTH_TIMEOUT`. Otherwise, the health-check request fails, prompting trouble management to perform the procedure defined in the trouble treatment table. (For more information about the trouble treatment table, see the *SINAP/SS7 User's Guide* (R8051).)

PARAMETERS

* `piblk` (input)

  Specifies a pointer to the `I_Block` that contains the health-check request. Use the `I_Block` pointed to by the `ca_get_msg` function's `piblk` parameter. The `I_Block` is stored in the `i_block_t` structure, which is described in the following section "The Main `I_Block` Structure (`i_block_t`)."

  The `I_Block` retrieved by `ca_get_msg()` is the `I_Block` that actually contains the health-check request.

Main `I_Block` Structure (`i_block_t`)

The following fields are set in the `i_block_t` structure, which is defined in the include file `iblock.h`. The `iblock.h` include file defines the structure of messages (`I_Blocks`) sent

via IPC. An `I_Block` is composed of a CASL control part, a transaction part, a timestamp, a node ID, an originator key, a destination key, and a message body.

```
typedef  struct  i_block_s
{
    ca_ctrl_t       ca_ctrl;
    ipc_trans_t     trans;
    timestamp_t     ts;
    node_id_t       node;
    ipc_key_t       orig_id;
    ipc_key_t       dest_id;
    ipc_data_t      msg;
} i_block_t;
```

* `ca_ctrl` (input)
  Specifies the CASL control structure for this `I_Block`. For more information about this structure, see "The CASL Control Structure (`ca_ctrl_t`)" later in this section.

* `trans` (input)
  Specifies the transaction ID structure. For information about this structure, see "The `I_Block` Transaction ID Structure (`ipc_trans_t`)" later in this section.

* `ts` (input)
  Specifies a collection of timestamps that the SINAP/SS7 system automatically inserts. The timestamps aid monitoring and logging, and are visible when you run the BITE log-analysis program. For information about this structure's fields, see "The Timestamp Structure (`timestamp_t`)" later in this section.

* `node` (input)
  Specifies the node ID. This structure is internal to the SINAP/SS7 system and should not be modified.

* `orig_id` (input)
  Specifies the `ipc_key_t` structure that contains the IPC key for a sender application process. For information about the `ipc_key_t` structure, see "The IPC Key Structure (`ipc_key_t`)" later in this section.

* `dest_id` (input)
  Specifies the `ipc_key_t` structure that contains the IPC key for the intended destination process of the `I_Block`. You can obtain this IPC key by calling the `ca_get_key()` function. For information about the `ipc_key_t` structure, see "The IPC Key Structure (`ipc_key_t`)" later in this section.

* `msg` (input)

  Specifies the `ipc_data_t` structure that contains the IPC user data. For information about the `ipc_data_t` structure, see "The IPC Data Structure (`ipc_data_t`)" later in this section.

CASL Control Structure (`ca_ctrl_t`)

The `ca_ctrl_t` structure contains the following fields and is defined in the include file `blkhdr.h`.

```
typedef struct   ca_ctrl_s
{
       int     msg_type;       /* For compatibility with the existing UNIX IPC
                                     mechanism. */

       int     msu_cnt;        /* # of MSUs pending */
       int     free_cnt;       /* # of free MSUs in read queue */
       int     wfree_cnt;      /* # of free MSUs in write queue */
       S16     lost_cnt;       /* # of MSUs lost due to insuff. resources */
       S16     data_size;      /* Total size of structure excluding this
                                  structure. */
       U8      node_index;     /* index (0 - 3) of current node */
       U8      sinap_variant;  /* V_CCITT, V_ANSI, V_HYBRID, V_TTC */
       U16     link;           /* index of the origination link */
       pid_t   pid;            /* Process ID of a specific process or 0
                                  for load distribution */
       int     msg_sender;     /* Set to 0 if from link otherwise contains
                                  the process ID */
       U8      iblk;           /* TRUE if data contains I_Block */
                               /* Not used anywhere!!!!! */
       U8      rw;             /* Flag for monitor message only */
       U8      monitor_id;     /* monitor ID for monitored MSU  */
       U8      ssn_sio;        /* SSN or SIO ID for load distribution. */
       struct {
               U32     timer_id;       /* For CASL internal use only */
               U32     timer_val;      /* For CASL internal use only */
               int     omsg_type;      /* For CASL internal use only */
       } timr;
       struct {                        /* For internal use only */
               int     source;         /* For distribution managment. */
               int     destination;    /* For protocol processing. */
#ifdef _KERNEL
               mblk_t  *mptr;          /* Back pointer to mblk_t.  L3 only.*/
#else
               void    *mptr;          /* ss7-1102 - dummy back pointer. */
#ifdef _LP_32_64_
               U32  filler;            /* For User32/Driver64 compatibility*/
#endif  /* _LP_32_64_ */
#endif  /* _KERNEL */
       } internal;
} ca_ctrl_t;
```

* `msg_type` (input)
  Specifies the type of MSU being sent. This field is compatible with the existing UNIX IPC mechanism.

* `data_size` (input)
  Specifies the total size of the structure, excluding this field.

* `node_index` (input)
  This field is internal to the SINAP/SS7 system and is automatically initialized to the appropriate value for the SINAP node being used.

* `sinap_variant` (input)
  This field is internal to the SINAP/SS7 system and is automatically initialized to the appropriate value for the network variant being used on the SINAP node.

* `lost_cnt` (input)
  Specifies the number of `M_Blocks` lost due to insufficient resources within the SS7 driver.

* `msu_cnt` (input)
  Specifies the number of MSUs pending in the READ queue.

* `free_cnt` (output)
  Specifies the number of free MSUs pending in the read queue.

* `wfree_cnt` (output)
  Specifies the number of free MSUs pending in the write queue.

The following fields are internal to the SINAP/SS7 system and you should not modify them:

* `pid` (input)
* `link` (input)
* `msg_sender` (input)
* `iblk` (input)
* `rw` (input)
* `monitor_id` (input)
* `ssn_sio` (input)
* `source` (input)
* `destination` (input)
* `mptr` (output) - Specifies a pointer to `m_block_t`, level 3.
* `timer_id` (input)
* `timer_val` (input)
* `omsg_type` (input)

IPC Transaction ID Structure (`ipc_trans_t`)

The following fields make up the `ipc_trans_t` structure, which is defined in the include file `iblock.h`.

```
typedef  struct  ipc_trans_s
{
         int     msg_type;
         U32     ref_nbr;
         U16     rw_ind;
         U8      monitor_id;
         U8      scenario_id;
} ipc_trans_t;
```

* `msg_type` (input)
  Specifies the basic message function identifier that the SINAP/SS7 system and client applications use to identify a message. When defining client application messages, you should specify message types within the range of `CL_IPC_MIN` and `CL_IPC_MAX` (see the include file `iblock.h` for more information).

* `ref_nbr` (input)
  Specifies a reference number that allows the sending and receiving processes to keep track of messages that are of the same type. The reference number lets the receiving process associate a reply with an instance of a command.

The following fields are internal to the SINAP/SS7 system and you should not modify them:

* `rw_ind` (input) - Specifies a read or write indicator for the monitor.

* `monitor_id` (input) -Associates the IPC message with a particular BITE monitor session.

* `scenario_id` (input) - Associates the IPC message with a particular BITE scenario execution session.

Timestamp Structure (`timestamp_t`)

The `timestamp_t` structure contains the following fields and is defined in the include file `timestamp.h`.

```
typedef struct timestamp_s
{
         U16     index;
         stamp_t stamp[MAX_TIME_STAMPS];
}timestamp_t;
```

* `index` (input)
  Specifies the next slot to stamp the time.

* `stamp[MAX_TIME_STAMPS]` (input)
  Specifies the timestamp slots. See "The `stamp_t` Structure" below for an explanation. (`MAX_TIME_STAMPS` is defined in the SINAP/SS7 `timestamp.h` include file.)

The `stamp_t` Structure

The `stamp_t` structure contains the following fields and is defined in the include file `timestamp.h`.

```
typedef struct stamp_s
{
       U32     secs;                   /* time in seconds since 1/1/70 */
       U8      tsid;                   /* timestamp id                 */
       U8      ipcx;                   /* ipc index if applicable      */
       U16     msec;                   /* time in milliseconds         */
} stamp_t;
```

* `secs` (input)
  Specifies the time (in seconds) since 1/1/70.

* `tsid` (input)
  Specifies the timestamp ID. Valid values are defined in the include file `timestamp.h`.

* `ipcx` (input)
  Specifies the IPC index, if applicable.

* `msec` (input)
  Specifies the time, in milliseconds.

The `node_id_t` Structure

The `node_id_t` structure contains the following fields and is defined in the include file `iblock.h`.

```
typedef struct node_id_s
{
        U8     ni;
        U32    spc;
} node_id_t;
```

* `ni` (input)
  Specifies the network indicator of the node. This field is internal to the SINAP/SS7 system and should not be modified.

* `spc` (input)
  Specifies the signaling point code of the node. This field is internal to the SINAP/SS7 system and should not be modified.

IPC Key Structure (`ipc_key_t`)

The `ipc_key_t` structure contains the following fields and is defined in the include file `sinap.h`.

```
typedef struct ipc_key_s
{
          U32      node;
          U32      module;
          U32      appl;
          U32      proc;
          U8       instance;
          U8       node_index;
          U16      ipc_index;
} ipc_key_t;
```

* `node` (output)
  Specifies the ID of the SINAP node on which your application is running. You can determine this value from the `NODE=` entry in the `/etc/sinap_master` file. You should modify any script files or user-defined program files that contain an invalid, hard-coded node name.

* `module` (output)
  Specifies the name or ID of the module. You can determine this value from the `MODULE=` entry in the `/etc/sinap_master` file. You should modify any script files or user-defined program files that contain an invalid, hard-coded module name.

* `appl` (input)
  Specifies the compressed application ID.

* `proc` (input)
  Specifies the compressed process ID.

* `instance` (input)
  Specifies the instance ID (in the range `1` through `8`). A value of `0` indicates that the field is not used.

* `node_index` (input)
  Specifies the index (0 through 3) of the node.

* `ipc_index` (input)
  Specifies the index ID of the IPC process table.

IPC Data Structure (`ipc_data_t`)

The following fields make up the `ipc_data_t` structure, which is defined in the include file `iblock.h`.

```
typedef  struct   ipc_data_s
{
         U8      more_ind;
         U32     len;
         U32     ret_code;
#if defined(__LP64__) || defined(_LP_32_64_)
         U32     filler; /* For User32/Driver64 compatibility */
#endif /* __LP64__ || _LP_32_64_ */
} ipc_data_t;
```

* `more_ind` (input)
  Specifies whether an IPC message is the last in a sequence. This field is useful if the data portion of a message exceeds the maximum amount of data that UNIX can send in a single data packet. Note that the limit is an UNIX configuration parameter, initially set to 4096 octets. The SINAP/SS7 system also uses `more_ind` when a command reply exceeds the response timeout. In this case, a command reply could consist of an arbitrary number of messages and a final reply; the `more_ind` field would be set to `1` to indicate that the receiving process is working on the reply. The final reply would indicate the result of the command.

* `len` (input)
  Specifies the length (in octets) of the data portion of the message body.

* `ret_code` (input)
  Specifies a return code value. By returning a user-defined value, a client application can use this field to indicate success or failure.

NOTE

The data portion of a message should follow the message field of `i_block_t`. The structure of the data portion is dependent

on the `msg_type` field. The following use of `i_block_t` is recommended.

```
typedef struct user_struc_s
{
  i_block_t   iblk_hdr;
  char        user_data[MAX_IBLK_DATA_SZ];
} user_struc_t;
```

FILES

`arch.h`, `ca_error.h`, `iblock.h`, `sinap.h`, `sysdefs.h`, `sys/time.h`, `timestamp.h`

RETURN VALUES

The `ca_health_chk_resp()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

Possible CASL values for `errno` are as follows.

Value	Meaning
`CA_ERR_ACCESS`	The calling process is not registered. Call `ca_register()` before calling this function.
`CA_ERR_IPC_KEY`	Invalid IPC key.

This function performs a `ca_put_msg()` and can also return the errors listed under that function.

SEE ALSO

`ca_health_chk_req()`, `ca_put_msg()`, `ca_register()`

ca_pack()

SYNOPSIS

```
U32 ca_pack(char s[])
```

INCLUDE FILES

```
$SINAP_HOME/Include/caslinc.h
$SINAP_HOME/Include/ca_glob.h
```

DESCRIPTION

The `ca_pack()` function converts a character string to a zero-filled, right-justified U32 word and returns the results. Typically, an application's name is defined as a character string; however, it must be defined as a zero-filled, right-justified U32 word in the `appl` field of the `dist_cmd_t` structure, which an application uses to implement enhanced message distribution. To convert the application name to the proper format, pass the `ca_pack()` function the character string that defines the application name. The function converts the string to a zero-filled, right-justified U32 word and returns the results, which can then be used in the `dist_cmd_t` structure's `appl` field.

RETURN VALUES

The `ca_pack()` function returns the application name as a zero-filled, right-justified U32 word.

NOTES

The man page format of this command is `ca_pack`.

ca_put_event()

SYNOPSIS

```
int  ca_put_event(
       U8         category,
       U8         subcategory,
       U8         type,
       int        state;
       int        code,
       ipc_key_t  *pipckey,
       char       *ptext);
```

DESCRIPTION

The `ca_put_event()` function sends an alarm or event to the Node Management subsystem.

PARAMETERS

* `category` (input)
  Specifies the classification of the set of events. A category can be established for any set of events sharing the same subcategories. Categories are defined on a system-wide basis; each value for the `category` parameter has a unique meaning to the system. Use any number in the range `15` through `30` for events that the client process generates. The values for this parameter are defined as constants in the include file `event.h`. (See the *SINAP/SS7 User's Guide* (R8051) for information about possible values for this parameter.)

* `subcategory` (input)
  Specifies the classification of an event within a category. The client processes sharing the event `category` must make the subcategory assignments and reservations. You can specify up to 30 subcategories. The values for this parameter are defined as constants in the include file `event.h`. (See the *SINAP/SS7 User's Guide* (R8051) for information about possible values for this parameter.)

* `type` (input)
  Specifies the classification of a particular event. Use `1` for a hardware event, `2` for a software event, and `3` for a network event. Hardware and network events are logged to the Alarm History log file. Software events are logged to the Software Notebook. The values for this parameter are defined as constants in the include file `event.h`.

* `state` (input)
  Specifies the state of the process at the time an error is detected.

* `code` (input)
  Specifies process-dependent code that provides additional information about the process at the time an event was detected.

* `pipckey` (input)
  Specifies a pointer to the `ipc_key_t` structure that contains the culprit IPC key. The IPC key's structure is defined in the include file `sinap.h`. For an explanation of the structure's fields and possible values, see "The IPC Key Structure (`ipc_key_t`)," which follows.

* `ptext` (input)
  Specifies a pointer to text associated with a particular event. The value of `ptext` can be up to 76 characters in length.

IPC Key Structure (`ipc_key_t`)

The `ipc_key_t` structure contains the following fields and is defined in the include file `sinap.h`.

```
typedef struct ipc_key_s
{
          U32      node;
          U32      module;
          U32      appl;
          U32      proc;
          U8       instance;
          U8       node_index;
          U16      ipc_index;
} ipc_key_t;
```

* `node` (output)
  Specifies the ID of the SINAP node on which your application is running. You can determine this value from the `NODE=` entry in the `/etc/sinap_master` file. You should modify any script files or user-defined program files that contain an invalid, hard-coded node name.

* `module` (output)
  Specifies the name or ID of the module. You can determine this value from the `MODULE=` entry in the `/etc/sinap_master` file. You should modify any script files or user-defined program files that contain an invalid, hard-coded module name.

* `appl` (input)
  Specifies the compressed application ID.

* `proc` (input)
  Specifies the compressed process ID.

* `instance` (input)
  Specifies the instance ID (in the range `1` to `8`). A value of `0` indicates that the field is not used.

* `node_index` (input)
  Specifies the index (0 through 3) of the node.

* `ipc_index` (input)
  Specifies the index ID of the IPC process table.

FILES

`arch.h`, `ca_error.h`, `event.h`

RETURN VALUES

The `ca_put_event()` function can return the following values. If the function returns `-1`, there is an error. See `ca_error.h` for the CASL error number and meaning; see `sys/errno.h` for UNIX errors.

Value	Meaning
`0`	Successful.
`-1`	Unsuccessful. See `errno` for error number and description.

Possible UNIX values for `errno` are as follows.

Value	Meaning
`EBADF`	An invalid open file descriptor was specified.
`ENOTTY`	This `fides` is not associated with a device driver that accepts control functions.
`EFAULT`	The pointer to the specified message is outside the address space allocated to the process.
`EINVAL`	Queue ID is not a valid message queue ID. The value of `msg_type` is less than `1`, or `msg_sz` is greater than `0` or the system-imposed limit.
`EIO`	An I/O error occurred during a read or write operation.

Value	Meaning
ENXIO	The requested service cannot be performed on this particular subdevice.
ENOLINK	The link to a requested machine is no longer active.

A possible CASL value for `errno` follows.

Value	Meaning
CA_ERR_ACCESS	The process calling `ca_put_event()` is not registered. Call `ca_register()` before calling this function.

This function performs a `ca_get_key()` and can also return the errors listed under that function.

ca_unpack()

SYNOPSIS

```
void from U32 ca_unpack(U32 value, char *ps);
```

INCLUDE FILES

```
$SINAP_HOME/Include/caslinc.h
$SINAP_HOME/Include/ca_glob.h
```

DESCRIPTION

The `ca_unpack()` function converts a zero-filled, right-justified U32 word to a character string. Typically, an application's name is defined as a character string; however, it must be defined as a zero-filled, right-justified U32 word in the `appl` field of the `dist_cmd_t` structure, which an application uses to implement enhanced message distribution. To convert the U32-word format of the application name back to a character string, call `ca_unpack()`, passing it the application name in the `value` parameter. The function writes the converted application name to the character string pointed to by the `*ps` parameter.

PARAMETERS

* `value` (input)
  Specifies the zero-filled, right-justified U32 word to be converted to a character string.

* `*ps` (input)
  Specifies a pointer to the character string in which `ca_unpack()` writes the results of the convariant.

NOTES

The man page format of this command is `ca_unpack`.

Appendix A
SINAP/SS7 MML Command Summary

The following chart lists and describes the SINAP/SS7 Man Machine Language (MML) commands and includes the alternate and UNIX online manual page (man page) command format for each command, if applicable. This chart is intended to serve as a quick reference guide to the MML commands. For detailed descriptions of how to execute commands using the Terminal Handler menus and/or the command line, and for descriptions of how to use the BITE subsystem, see the *SINAP/SS7 User's Guide* (R8051).

The offline Built-In Test Environment (BITE) Log Analysis commands and `sy` utility commands are listed at the end of the chart. These commands can only be run from the command line, not from the Terminal Handler.

Table A-1. MML Command Summary *(Page 1 of 29)*

MML Command	Description	Alternate/ Man Page Command
`BACKUP-APPL`	Backup the application (copy from the source to the destination storage device). Terminal Handler menu: Application Commands Format: `BACKUP-APPL:SOURCE=source, DESTINATION=Destination;`	`BKUP-APPL`
`BACKUP-NODE`	Backup the node (backup the existing node static database to disk). Terminal Handler menu: System Commands Format: `BACKUP-NODE;`	`BKUP-NODE`
`CHANGE-BKUPDAY`	Change the backup day (change the number of days for automatic node backup). Terminal Handler menu: System Commands Format: `CHANGE-BKUPDAY:DAYS=days;`	`CHG-BKDAY`

Table A-1. MML Command Summary *(Page 2 of 29)*

MML Command	Description	Alternate/ Man Page Command
CHANGE-CLSET	Change the combined link set. Terminal Handler menu: CHANGE command on Network Commands menu Format: CHANGE-CLSET:LINKSET=*linkset*,EMERGENCY=*yes*;	CHG-CLSET
CHANGE-CPC	Change the concerned point code (CPC) (add or delete an individual remote point code). Terminal Handler menu: CHANGE command on Network Commands menu Format: CHANGE-CPC:LSSN=ssn,ADDRPC=pc[&pc]; CHANGE-CPC:LSSN=ssn,DELRPC=pc[&pc];	CHG-CPC
CHANGE-DUCPC (Not supported in TTC network variant)	Change the duplicate concerned point code (DUCPC). Terminal Handler menu: CHANGE command on Network Commands menu Format: CHANGE DUCPC:LSSN=ssn,NEWRPC=pc;	CHG-DUCPC
CHANGE-GTT	Change the global title (replace an existing global title entry with a new one). The user must select at least one of the three fields. The number of fields in this command changes with the values entered. (The HADDR field is optional.) Refer to the section on global title format in Chapter 3 for more information on global title values.) Terminal Handler menu: CHANGE command on Network Commands menu Format: CHANGE-GTT:OLD_GTI=gti,OLD_TT=tt,OLD_NP=np ,OLD_NOAI=noai, OLD_LADDR=laddr,GTI=gti,TT=tt, NP= np,NOAI=noai,LADDR=laddr [,HADDR=haddr] {,DPC=dpc | ,SSN=ssn| ,NADDR=naddr}[SSN2=ssn2][DPC2=dpc2];	CHG-GTT

Table A-1. MML Command Summary *(Page 3 of 29)*

MML Command	Description	Alternate/ Man Page Command
CHANGE-LINK	Change a link. Terminal Handler menu: CHANGE command on Network Commands menu Format: CHANGE-LINK:LINK=*pc,*PERIODIC_SLT= *slt*; For the TTC network variant, use the command format: CHANGE-LINK:LINK=*pc,* PERIODIC_SLT= *srt*;	CHG-LINK
CHANGE-LSET	Change the link set (emergency alignment flag). Terminal Handler menu: CHANGE command on Network Commands menu Format: CHANGE-LSET:LINKSET=linkset, EMERGENCY=emergency;	CHG-LSET
CHANGE-PURGEDAY	Change the purgeday (the number of days the log file remains on disk before it is deleted). Terminal Handler menu: System Commands Format: CHANGE-PURGEDAY:LOG=logfile,DAYS=days;	CHG-PDAY
CHANGE-REMSSN	Change the remote subsystem number (SSN) (add or delete remote SSNs for a specified point code). Terminal Handler menu: CHANGE command on Network Commands menu Format: CHANGE-REMSSN:PC=pc,ADDSSN=ssn; CHANGE-REMSSN:PC=pc,DELSSN=ssn;	CHG-REMSSN

Table A-1. MML Command Summary *(Page 4 of 29)*

MML Command	Description	Alternate/ Man Page Command
CHANGE-RSET	Change the route set (add, delete, or exchange the priority of a route; enable or disable load sharing between two route sets). Terminal Handler menu: CHANGE command on Network Commands menu Format: CHANGE-RSET:ROUTESET=routeset,ADDROUTE=route,PRIORITY=priority; CHANGE-RSET:ROUTESET=routeset,DELROUTE=route;CHANGE-RSET:ROUTESET=routeset,XROUTE1=route,XROUTE2=route; CHANGE-RSET:ROUTESET=routeset,LOADSHR=loadshr;	CHG-RST
CHANGE-SYSTAB	Change the system table timers and thresholds. Terminal Handler menu: CHANGE command on Network Commands menu Format: CHANGE-SYSTAB:TABID=tabid,TIMER=timer,NEWTIME=time; CHANGE-SYSTAB:TABID=tabid,THRESHOLD=threshold,NEWLEVEL=level;	CHG-SYSTAB
CONFIGURE-LINK	Configure a link (activate or deactivate a link state). Terminal Handler menu: CONFIGURE command on Network Commands menu Format: CONFIGURE-LINK;LINK=link,STATE=state;	CFR-LINK
CONFIGURE-LSET	Configure a link set (activate or deactivate a link set). Terminal Handler menu: CONFIGURE command on Network Commands menu Format: CONFIGURE-LSET:LINKSET=linkset,STATE=state;	CFR-LSET

Table A-1. MML Command Summary *(Page 5 of 29)*

MML Command	Description	Alternate/ Man Page Command
CONFIGURE-RSET	Configure the route set (activate/deactivate a route set). Terminal Handler menu: CONFIGURE command on Network Commands menu Format: CONFIGURE-RSET:ROUTESET=routeset, STATE=state;	CFR-RSET
CHANGE-SLSTYPE (ANSI network variant only)	Change the Signaling Link Selection (SLS) type (5-bit or 8-bit) used for all incoming and outgoing traffic. Terminal Handler menu: CHANGE command on Network Commands menu Format:CHANGE-SLSTYPE:TYPE=numeric_slstype; Were numeric_slstype is 8 or 5.	CHG-SLSTYPE
CREATE-CLSET (ANSI network variant only)	Create a combined link set (that is, two link sets that communicate with a mated pair of signaling transfer points [STPs]). Terminal Handler menu: CREATE command on Network Commands menu Format:CREATE-CLSET:CLSET=clset,LSET1= linkset1,LSET2=linkset2;	CRTE-CLSET
CREATE-CPC	Create Concerned Point Code (associate a remote point code with a local subsystem). Terminal Handler menu: CREATE command on Network Commands menu Format: CREATE-CPC:LSSN=ssn,RPC=pc[&pc];	CRTE-CPC
CREATE-DUCPC (Not supported in TTC network variant)	Create a duplicate concerned point code (assign one of the concerned point codes as the replicate for a specified local subsystem). Terminal Handler menu: CREATE command on Network Commands menu Format: CREATE-DUCPC:LSSN=ssn, RPC=pc;	CRTE-DUCPC

Table A-1. MML Command Summary *(Page 6 of 29)*

MML Command	Description	Alternate/ Man Page Command
CREATE-FOPC (ANSI network variant only)	Create a fictitious originating point code to be used in place of the calling party's originating point code [OPC]. Terminal Handler menu: CREATE command on Network Commands menu Format: CREATE-FOPC:FOPC=network-cluster-member;	CRTE-FOPC
CREATE-GTT	Create a global title translation (define a duplicate point code [DPC], subsystem number [SSN], and/or address information that replaces the original global title entry.) Terminal Handler menu: CREATE command on Network Commands menu Format: CREATE-GTT:GTI=1,NOAI=noai,LADDR=laddr [,HADDR=haddr]{,DPC=dpc|,SSN=ssn|, NADDR=naddr}; CREATE-GTT:GTI=2,TT=tt, LADDR=laddr, [, HADDR=haddr]{,DPC=dpc|,SSN=ssn|, NADDR=naddr};	CRTE-GTT
CREATE-GTT (cont.)	CREATE-GTT:GTI=3, TT=tt,NP=np, LADDR=laddr,[,HADDR=haddr] {,DPC=dpc|,SSN=ssn|,NADDR=naddr}; CREATE-GTT:GTI=4, TT=tt, NP=np,NOAI=noai,LADDR=laddr, [,HADDR=haddr]{,DPC=dpc|,SSN=ssn|, NADDR=naddr};	
CREATE-LINK	Create a link. Terminal Handler menu: CREATE command on Network Commands menu Format: CREATE-LINK:LINK=link,PORTNUM= portnum,LINKSET=linkset, SLC=slc,PRIORITY=priority,SPEED=speed;	CRTE-LINK

Table A-1. MML Command Summary *(Page 7 of 29)*

MML Command	Description	Alternate/ Man Page Command
CREATE-LSET	Create a link set. Terminal Handler menu: CREATE command on Network Commands menu Format for the CCITT network variant: CREATE-LSET:LINKSET=linkset,ADPC=adpc, LOADLINK=loadlink, ACTLINK=actlink; Format for the ANSI network variant: CREATE-LSET:LINKSET=linkset,ADPC=adpc, TYPE=type,LOADLINK=loadlink,ACTLINK= actlink;	CRTE-LSET
CREATE-OSP	Create the own signaling point code (for specified network). Terminal Handler menu: CREATE command on Network Commands menu Format: CREATE-OSP:NETWORK=network,SPC=spc;	CRTE-OSP
CREATE-REMSSN	Create Remote Subsystem. Terminal Handler menu: CREATE command on Network Commands menu Format: CREATE-REMSSN:PC=pc,SSN=ssn[&ssn];	CRTE-REMSSN
CREATE-RSET	Create a route set. Terminal Handler menu: CREATE command on Network Commands menu Format: CREATE-RSET:ROUTESET=routeset, DPC=dpc,ROUTES=routes[&routes],LOADSHR= loadshr;	CRTE-RSET
DELETE-CLSET (ANSI network variant only)	Delete the combined link set. Terminal Handler menu: DELETE command on Network Commands menu Format: DELETE-CLSET:CLSET=clset;	DLT-CLSET

Table A-1. MML Command Summary *(Page 8 of 29)*

MML Command	Description	Alternate/ Man Page Command
DELETE-CPC	Delete the concerned point code (CPC) (delete *all* remote point codes for specified local subsystem). Use CHANGE-CPC to delete an individual concerned point code (CPC). Terminal Handler menu: DELETE command on Network Commands menu. Format: DELETE-CPC:LSSN=ssn;	DLT-CPC
DELETE-DUCPC (Not supported in TTC network variant)	Delete the duplicate concerned point code (DUCPC). Terminal Handler menu: DELETE command on Network Commands menu. Format: DELETE-DUCPC:LSSN=ssn;	DLT-DUCPC
DELETE-FILE	Delete a file from disk. Terminal Handler menu: System Commands. Format: DELETE-FILE:FILE=filename;	DLT-FILE
DELETE-FOPC (ANSI network variant only)	Delete a fictitious originating point code. Terminal Handler menu: DELETE command on Network Commands menu Format: DELETE-FOPC:FOPC=network-cluster-member;	DLT-FOPC

Table A-1. MML Command Summary *(Page 9 of 29)*

MML Command	Description	Alternate/ Man Page Command
DELETE-GTT	Delete a global title translation (GTT). See "Global Title Formats" in Chapter 3 for an explanation of the values for this command. Terminal Handler menu: DELETE command on Network Commands menu Format for the CCITT, China, and TTC variants: DELETE-GTT:GTI=1,NOAI=noai,LADDR= laddr; DELETE-GTT:GTI=2,TT=tt,LADDR=laddr; DELETE-GTT:GTI=3,TT=tt,NP=np,LADDR= laddr; DELETE-GTT:GTI=4,TT=tt,NP=np,NOAI= noai,LADDR=laddr; Format for the ANSI variant: DELETE-GTT:GTI=1,TT=tt,NP=np,LADDR= laddr; DELETE-GTT:GTI=2,TT=tt,LADDR=laddr;	DLT-GTT
DELETE-LINK	Delete a link. Terminal Handler menu: DELETE command on Network Commands menu Format: DELETE-LINK: LINK=link;	DLT-LINK
DELETE-LSET	Delete a link set. Terminal Handler menu: DELETE command on Network Commands menu Format: DELETE-LINK: LINK=link;	DLT-LSET
DELETE-OSP	Delete an own signaling point code (OSP) from the network (all other network elements must already be deleted). Terminal Handler menu: DELETE command on Network Commands menu Format: DELETE-OSP;	DLT-OSP

Table A-1. MML Command Summary *(Page 10 of 29)*

MML Command	Description	Alternate/ Man Page Command
DELETE-REMSSN	Delete a remote subsystem number (SSN) (stop monitoring for all subsystems at a specified remote node). Terminal Handler menu: DELETE command on Network Commands menu Format: DELETE-REMSSN:RPC=pc;	DLT-REMSSN
DELETE-RSET	Delete a route set. Terminal Handler menu: DELETE command on Network Commands menu Format: DELETE-RSET:ROUTESET=routeset;	DLT-RSET
DISABLE-LOAD-CONTROL	Disable load control (deactivate load control processing). Terminal Handler menu: DELETE command on Load Control menu Format: DISABLE-LOAD-CONTROL:SSN=ssn [,INSTANCE=[&instance]];	DISABLE-LC
DISPLAY-BKUPDAY	Display the backup day (number of days in node database backup cycle). Terminal Handler menu: DISPLAY command on System Commands menu Format: DISPLAY-BKUPDAY;	DISPL-BKDAY
DISPLAY-CLSET (ANSI network variant only)	Display the combined link set. Terminal Handler menu: DISPLAY command on System Commands menu Format: DISPLAY-CLSET:CLSET=clset[,PRINT=print];	DISPL-CLSET

Table A-1. MML Command Summary *(Page 11 of 29)*

MML Command	Description	Alternate/ Man Page Command
DISPLAY-CPC	Display the concerned point code (CPC) for a local subsystem. Terminal Handler menu: DISPLAY command on System Commands menu Format: DISPLAY-CPC:SSN=ssn[,PRINT=print];	DISPL-CPC
DISPLAY-FOPC (ANSI network variant only)	Display the fictitious originating point code. Terminal Handler menu: DISPLAY command on System Commands menu Format: DISPLAY-FOPC:FOPC=network-cluster-member;	DISPL-FOPC
DISPLAY-GTT	Display the global titles. Terminal Handler menu: DISPLAY command on System Commands menu Format: DISPLAY-GTT;	DISPL-GTT
DISPLAY-LINK	Display a link. Terminal Handler menu: DISPLAY command on System Commands menu Format: DISPLAY-LINK:LINK=link[,PRINT=print];	DISPL-LINK
DISPLAY-LSET	DIsplay a link set. Terminal Handler menu: DISPLAY command on System Commands menu Format: DISPLAY-LSET:LINKSET=linkset [,PRINT=print];	DISPL-LSET

Table A-1. MML Command Summary *(Page 12 of 29)*

MML Command	Description	Alternate/ Man Page Command
DISPLAY-LOAD-CONTROL	Display load control statistics for a specified application. Terminal Handler menu: DISPLAY command on Load Control menu Format: DISPLAY-LOAD-CONTROL:SSN=ssn [,PRINT=print];	
DISPLAY-MON	Display active BITE monitor IDs. Terminal Handler menu: BITE Commands Format: DISPLAY-MON;	DISPL-MON
DISPLAY-OSP	Display the own signaling point code (OSP). Terminal Handler menu: DISPLAY command on System Commands menu Format: DISPLAY-OSP:[PRINT=print];	DISPL-OSP
DISPLAY-PROCESS-VERSION	Display the version of a process. This command is available *only* through the Terminal Handler BITE COMMANDS menu.	DISPL-PVERS
DISPLAY-PURGEDAY	Display the purgeday (the number of days a log file remains on disk before it is deleted). Terminal Handler menu: System Commands Format: DISPLAY-PURGEDAY:LOG=logfile;	DISPL-PDAY
DISPLAY-REMSSN	Display the remote subsystem numbers (SSN) for a specified remote point code). Terminal Handler menu: DISPLAY command on System Commands menu Format: DISPLAY-REMSSN:PC=pc[,PRINT=print];	DISPL-REMSSN

Table A-1. MML Command Summary *(Page 13 of 29)*

MML Command	Description	Alternate/ Man Page Command
DISPLAY-RSET	Display the route set. Terminal Handler menu: DISPLAY command on System Commands menu Format: DISPLAY-RSET:DPC=dpc[,PRINT=print]; DISPLAY-RSET:ROUTESET=routeset[,PRINT=print];	DISPL-RSET
DISPLAY-SCEN	Display the scenario (that is, the current active scenario with scenario ID BITE assigned to it). Terminal Handler menu: SCENARIO command on the BITE Commands menu. Format: DISPLAY-SCEN;	DISPL-SCEN
DISPLAY-SUBSYSTEM	Display the current status of the subsystem. Terminal Handler menu: Application Commands Format: DISPLAY-SUBSYSTEM:SSN=ssn;	DISPL-SUBSYS
DISPLAY-SYSTAB	Display the system table for Message Transfer Part [MTP] and Signaling Connection Control Point [SCCP] timer values and MTP threshold values. Terminal Handler menu: DISPLAY command on System Commands menu Format:DISPLAY-SYSTAB;TABID=tabid, TIMER=timer[,PRINT=print]; DISPLAY-SYSTAB;TABID=tabid,THRESHOLD=threshold[,PRINT=print];	DISPL-SYSTAB
DUMP-TABLE	Dump the contents of the MTP routing and management tables to the static table file in binary format. Terminal Handler menu: Network Commands Format: DUMP-TABLE;	DUMP-TABLE

Table A-1. MML Command Summary *(Page 14 of 29)*

MML Command	Description	Alternate/ Man Page Command
ENABLE-LOAD-CONTROL	Enable load control (automatically activate load control processing for specified application when overload conditions occur). Terminal Handler menu: DISPLAY command on Load Control menu Format: ENABLE-LOAD-CONTROL:SSN=ssn [,INSTANCE=[&instance]];	
EXIT-LOAD-CONTROL	Exit load control (deactivate load control processing for specified application). This command can be used only if load control processing was initiated using INVOKE-LOAD-CONTROL. Terminal Handler menu: EXIT option on the Load Control menu Format:EXIT-LOAD-CONTROl:SSN=ssn [,INSTANCE=[&instance]];	
INVOKE-LOAD-CONTROL	Invoke load control processing for a specified application even if no overload condition exists. Terminal Handler menu: INVOKE command on the Load Control menu Format: INVOKE-LOAD-CONTROL:SSN=ssn [,INSTANCE=[&instance]];	
LPCR_cmd	When the DLPC feature is configured on the SINAP node, this command manipulates a specified remote point code to change its status from active to standby (or from standby to active), reset its circuit status, or dump the circuit states to a file. Issue the command from a SINAP login window. Format: LPCR_cmd -r <pc> [-asRDV -i instance]	LPC_cmd

Table A-1. MML Command Summary *(Page 15 of 29)*

MML Command	Description	Alternate/ Man Page Command
READ-TREAT	Read the file containing the trouble Treatment Table. Terminal Handler menu: System Commands Format: READ-TREAT;	READ-TREAT
REPORT-ALARM	Report the contents of the Alarm History file. Terminal Handler menu: System Commands Format: REPORT-ALARM:DATE=date[,PRINT=print];	RPT-ALARM
REPORT-MALL	Report MTP, SCCP, and TCAP subsystem measurements for a specified time period. Terminal Handler menu: 1. MEASUREMENTS command on Network Commands menu 2. REPORT MEASUREMENTS command on Measurement Commands menu Format: REPORT-MALL: DATE=CCYY-MM-DD,TIME=HH:MM, DATE=CCYY-MM-DD,TIME=HH:MM,FILE=file [,PRINT=print]; **Note:** For CCYY-MM-DD, CC=century (19 or 20), YY=year (38 through 99), MM=month (1 through 12), DD=day (1 through 31). For HH:MM, HH=hour (1 through 24), MM=minutes (00 through 59).	RPT-MALL

Table A-1. MML Command Summary *(Page 16 of 29)*

MML Command	Description	Alternate/ Man Page Command
REPORT-MMTP	Report MTP measurements for a specified time period. Terminal Handler menu: 1. MEASUREMENTS command on Network Commands menu 2. REPORT MEASUREMENTS command on Measurement Commands menu Format: REPORT-MMTP:DATE=CCYY-MM-DD,TIME=HH:MM, DATE=CCYY-MM-DD,TIME=HH:MM,FILE=file [,PRINT=print]; **Note:** For CCYY-MM-DD, CC=century (19 or 20), YY=year (38 through 99), MM=month (1 through 12), DD=day (1 through 31). For HH:MM, HH=hour (1 through 24), MM=minutes (00 through 59).	RPT-MMTP
REPORT-MSCCP	Report SCCP subsystem measurements for a specified time period. Terminal Handler menu: 1. MEASUREMENTS command on Network Commands menu 2. REPORT MEASUREMENTS command on Measurement Commands menu Format: REPORT-MSCCP:DATE=CCYY-MM-DD,TIME=HH:MM, DATE=CCYY-MM-DD,TIME=HH:MM,FILE=file [,PRINT=print]; **Note:** For CCYY-MM-DD, CC=century (19 or 20), YY=year (38 through 99), MM=month (1 through 12), DD=day (1 through 31). For HH:MM, HH=hour (1 through 24), MM=minutes (00 through 59).	RPT-MSCCP

Table A-1. MML Command Summary *(Page 17 of 29)*

MML Command	Description	Alternate/ Man Page Command
REPORT-MTCAP	Report TCAP subsystem measurements for a specified time period time. Terminal Handler menu: 1. MEASUREMENTS command on Network Commands menu 2. REPORT MEASUREMENTS command on Measurements Commands menu. Format: REPORT-MTCAP:DATE=CCYY-MM-DD,TIME=HH:MM,DATE=CCYY-MM-DD,TIME=HH:MM,FILE=file [,PRINT=print]; **Note:** For CCYY-MM-DD, CC=century (19 or 20), YY=year (38 through 99), MM=month (1 through 12), DD=day (1 through 31). For HH:MM, HH=hour (1 through 24), MM=minutes (00 through 59).	RPT-MTCAP
REPORT-NBOOK	Report the contents of the Software Notebook. Terminal Handler menu: System Commands Format: REPORT-NBOOK:DATE=date,[,PRINT=print];	RPT-NBOOK
RESTORE-APPL	Restore an application database from disk or tape. Terminal Handler menu: Application Commands Format: RESTORE-APPL:SOURCE=source, DESTINATION=destination;	RST-APPL
RESTORE-NODE	Restore node static database (primary or secondary copy) from disk. Terminal Handler menu: System Commands Format: RESTORE-NODE: FROM=source,RCLOG=rclog;	RST-NODE

Table A-1. MML Command Summary *(Page 18 of 29)*

MML Command	Description	Alternate/ Man Page Command
RETRIEVE-NOM	Retrieve and display the oldest 30-minute Node Network Management Measurement report. Terminal Handler menu: 1. MEASUREMENTS command on Network Commands menu 2. RETRIEVE MEASUREMENTS command on Measurement Commands menu. Format: RETRIEVE-NOM;	RTRV-NOM
RETRIEVE-SMR	Retrieve and display the most recent 5-minute Node Network Management Measurement report. Terminal Handler menu: 1. MEASUREMENTS command on Network Commands menu 2. RETRIEVE MEASUREMENTS command on Measurement Commands menu Format: RETRIEVE-SMR;	RTRV-SMR
SETUP-LOAD-CONTROL	Set up load control (define an application's load control threshold and operating characteristics). Terminal Handler menu: SETUP command on Load Control menu Format: SETUP-LOAD-CONTROL:SSN=ssn, TYPE=type,THRESHOLD=threshold, DELAY=delay,COUNT=count,ABATEDELAY= abatedelay;	SETUP-LC

Table A-1. MML Command Summary *(Page 19 of 29)*

MML Command	Description	Alternate/ Man Page Command

`send_cm`	Sends an MML command to the Command Management Process (`nmcm`) for execution. Allows you to execute MML commands from a SINAP login window (at the UNIX command level). Use this command to execute MML commands interactively, to execute a script file, or to issue single MML commands.
	
	Note that the `send_cm` command does not support the use of BITE MML commands.
	
	Format: `send_cm` (interactive mode; accepts commands via keyboard input)
	`send_cm file_name`
	`send_cm -s "mml_command;"`
		`send_cm`

`SET-PRINTER`	Prints all commands and responses to a terminal or specified printer. Terminal Handler menu: System Commands Format: `SET-PRINTER:PRINT=print;`	`SET-PRINTER`
`sinap_update`	Used to update link configuration information without rebooting the SINAP/SS7 system.	
`START-DBG`	Send a debug message to a specified process. Terminal Handler menu: BITE Commands Format: `START-DBG:ENT=(entity),MSG=message;`	`STA-DBG`
`START-MEASURE`	Start on-demand measurements for signaling information field (SIF) and service information octets (SIO) transmitted or received. Terminal Handler menu: `MEASUREMENTS` command on the Network Commands menu Format: `START-MEASURE:MEASURE=measure;`	`STA-MEAS`

Table A-1. MML Command Summary *(Page 20 of 29)*

MML Command	Description	Alternate/ Man Page Command
START-MON	Start the BITE monitor for specified entities from any SINAP process. Terminal Handler menu: MONITOR command on the BITE Commands menu Format: START-MON:ENT=(entity)[,DISP=Y/N] [,LOG=filename [(size)] [,CONT=Y/N]];	STA-MON
START-MWRITE	Start writing measurements to the measurement logs in the logs/system directory. Terminal Handler menu: MEASUREMENTS command on Network Commands menu Format: START-MWRITE;	STA-MWRITE
START-SCEN	Start a BITE scenario execution (network simulation). Terminal Handler menu: SCENARIO command on BITE Commands menu Format: START-SCEN:ENT=(entity),FILE=filename;	STA-SCEN

Table A-1. MML Command Summary *(Page 21 of 29)*

MML Command	Description	Alternate/ Man Page Command

`start_sinap`	Execute the `start_sinap` script file that starts the SINAP/SS7 system on each node to be activated.
	
	Issue this command from a UNIX operating system prompt and specify one of three start up modes:
	
	1) Verbose - Displays information on the terminal as commands execute
	
	2) Test-Environment - Displays information on the terminal as processes execute
	
	3) Test-Environment and Verbose - Combines test-environment and verbose modes to display (on the terminal) commands and processes as they execute
	
	Format: `start_sinap`
	`start_sinap -v`
	`start_sinap -t`
	`start_sinap -tv`
		`start_sinap`

`static2mml`	Save and re-create an existing SINAP/SS7 configuration in a specified input file (typically `STATIC_load`) and write the command output to a specified output. Issue this command at the UNIX operating system prompt. Format: `static2mml$SINAP_HOME/Bin/shm/pri/input_file [ > output_file]`	`static2mml`
`STOP-MEASURE`	Stop on-demand measurements (SIF and SIO octets transmitted or SIF and SIO octets received). Terminal Handler menu: `MEASUREMENTS` command on Network Commands menu Format: `STOP-MEASURE:Measure=measure;`	`STOP-MEAS`

Table A-1. MML Command Summary *(Page 22 of 29)*

MML Command	Description	Alternate/ Man Page Command
`STOP-MON`	Stop the BITE monitor for specified entity. Terminal Handler menu: `MONITOR` command on BITE Commands menu Format: `STOP-MON:ENT=monitor ID;`	`STOP-MON`
`STOP-MWRITE`	Stop writing measurements to the measurement logs in Logs/system. Terminal Handler menu: `MEASUREMENTS` command on Network Commands menu Format: `STOP-MWRITE;`	`STOP-MWRITE`
`STOP-SCEN`	Stop a specified BITE scenario execution. Terminal Handler menu: `SCENARIO` command on BITE Commands menu Format: `STOP-SCEN:ENT=scenario_id;`	`STOP-SCEN`
`stop_sinap`	Run the `stop_sinap` script file that stops the SINAP/SS7 system on each node to be stopped. Issue this command from a UNIX operating system prompt. Format: `stop_sinap`	`stop_sinap`
`TEST-LINK`	Send a signaling link test message (SLTM) to a specified link. (TTC and NTT send signaling route test (SRT) messages) Terminal Handler menu: BITE Commands Format: `TEST-LINK:LINK=link;`	`TEST-LINK`

Table A-1. MML Command Summary *(Page 23 of 29)*

MML Command	Description	Alternate/ Man Page Command
TEST-ROUTE	Perform a signaling route test for a specified route. (NTT and TTC only.) For additional information see the man page for TEST-ROUTE. Terminal Handler menu: BITE commands. Format: TEST-ROUTE:DPC=dpc,AB=ab; Note that ab corresponds to the A/B indicator in Q.707.	
BITE Log Analysis Commands These commands can only be run from the command line. (Before using these commands, start the BITE Log Analysis (bila) program.)		
bila	Starts the offline BITE Log Analysis program. Format: bila	bila
bidb	Starts the Database Builder program which is a menu-driven interface used to build different types of MSU messages for different types of applications and scenarios. The bidb is used to construct a test MSU for the scenario-execution application to send to the test application. Use the *message_file* option to specify the path name of a file to which an existing MSU has already been saved. Format: bidb*[message-file]*	bidb
DISPLAY:FILE	Display records from a specified BITE log file. Format: DISPLAY:FILE=logfile;	BDISPLAY
FIND:FILE	Extracts records from a specified BITE log file that satisfies **all** criteria specified within the command arguments. You can specify multiple keys and key values. Format: FIND:FILE=logfile, OFILE= file,key=key_value;	BFIND

Table A-1. MML Command Summary *(Page 24 of 29)*

MML Command	Description	Alternate/ Man Page Command
`SELECT:FILE`	Extracts records from a specified BITE log file that satisfies **any** criteria specified within the command arguments. You can specify multiple keys and key values. Format: `SELECT:FILE=logfile, OFILE= file,key=key_value;`	`BSELECT`
`SUMMARY:FILE`	Counts records in a specified BITE log file that satisfies the criteria specified within the command arguments. You can specify multiple keys and key values. Format: `SUMMARY:FILE=logfile, OFILE= file,key=key_value`	`BSUMMARY`
`QUIT`	Exits the BITE Log Analysis program. Formats: `QUIT or QUIT:; or QUIT:`	`BQUIT`
Frequently-Used sy Utility Commands		
`?`	Displays all available commands for the `sy` utility.	
`#APPL`	Displays application tables.	
`#BI,MD`*`x`*	Displays the BITE monitor table entry. Options for *x* include: • *0* - Display all. • *aname,pname,text* - Send the text to the `aname` or `pname` process. *text* = *"TRACE[,n]"* for trace dump for last *n* (specify number of events) events.	
`#DIST,`*`x`*	Displays distribution information. Substitute *optional appl_id* for *x* to display distribution information for a specific application table.	
`#IPC,`*`x`*	Checks the status of an application. This command displays IPC process table entry *x*. Use *x=0* to list all currently-running processes that are registered with the SINAP/SS7 system. For *x*, indicate the table entry to list. For example, if you enter the command, `#IPC,3`, you will see a list of the third entry in the interprocess communications (IPC) table.	

Table A-1. MML Command Summary *(Page 25 of 29)*

MML Command	Description	Alternate/ Man Page Command
`#IPC,LPC`	When the DLPC feature is configured on the SINAP node, the command lists all registered logical point codes (LPCs). If DLPC is not configured, the command produces the error message `This node is not provisioned for DLPC capabilities`. Issue the command from a SINAP login window. Format: `#IPC,LPC`	`#IPC,LPC`
`#IRT`	Displays the inbound routing table.	
`#KEY,aname[,pname[,inst]]`	Displays an IPC key table.	
`#lc,x`	Displays load control information. Options for *x* include *SSN_number* to display the information for a specific subsystem, or *0* to display information for all SSNs.	
`#LCD,SSN_number`	Displays load control debug information for a specific SSN.	
`#L3,CLS`	Displays MTP shared data for all combined linksets.	
`#L3,CLS,x`	Displays MTP shared data for a specific combined linkset *x*.	

Table A-1. MML Command Summary *(Page 26 of 29)*

MML Command	Description	Alternate/ Man Page Command
#L3,*x*	Displays MTP shared data. Options for *x* include: • dt - Displays the discrimination table entries. • lst*[,optional linkset_#]* - Displays timer values and includes information about the MTP restart timers: L3T19, L3T20, and L3T21. Displays values for the ANSI network variant timers: L3T22, L3T23, L3T24, L3T25, L3T26, L3T27, L3T28, L3T29, and L2T30. In addition to displaying timer information, the command: #L3,lst displays a flag for each active link set. The flag indicates whether the node connected by the link set is executing MTP restart. • port*[,optional linkset_#]*- Displays all ports or the port for a specified link set number. • res - Displays MTP restart information, such as the own_sp_restarting flag (which indicates whether this SINAP/SS7 node is currently executing MTP restart), the number of active link sets, the number of TRA messages received, and the L3T20 timer ID. • rst*[,optional n-c-m DPC_#]* - Displays Routeset information, such as the accessibility and availability of the DPC. • spf - Displays signaling point routing failure reports. • tim - Displays timer values and includes information about the MTP restart timers L3T19, L3T20, and L3T21. In addition, displays values for the ANSI network variant timers: L3T22, L3T23, L3T24, L3T25, L3T26, L3T27, L3T28, L3T29, and L2T30.	
#L3,RC	Traces delivery of internal messages within the MTP layers. When the DLPC feature is configured on the SINAP node, the command traces communications between MTP Level 3 routing control (L3RC) and the ISUP manager.	#L3,RC

Table A-1. MML Command Summary *(Page 27 of 29)*

MML Command	Description	Alternate/ Man Page Command
#ORT,CLS	Displays the outbound routing combined link set table. If you selected the 8-bit signaling link selection (SLS) processing scheme using the CHANGE-SLSTYPE command, then combined link set signaling link code (SLC) values are also displayed. If you did not select 8-bit SLS processing, the 5-bit SLS SLC values are displayed.	
#ORT,LS	Displays the outbound routing link set table. For the ANSI network variant, if you select the 8-bit signaling link selection (SLS) processing scheme using the CHANGE-SLSTYPE command, the 8-bit combined link set signaling link code (SLC) values are also displayed. If you did not select 8-bit SLS processing, then 5-bit SLS SLC values are displayed for link sets.	
#ORT,{RS or DPC#}	Displays the outbound routing signaling route set test signal (RST) table for a specified route set or duplicate point code.	
Q	Close the sy utility and quit all operations.	
#sc,*x*,*n*	Displays SCCP shared memory information. Options for *x* include: • appl*[,optional local SSN_number]* - Displays all application tables or the application table for a specified local SSN. • cpc*[,optional local SSN_number]* - Displays all concerned point codes (CPCs) or the CPCs for a specified local SSN. • dump - Dumps all SCCP shared memory information. • lrm - Displays the local reference memory (LRM) and the command's output. • lrn*[,optional LRN_number]* - Displays information about all active LRNs or the information for a specific LRN (n). • SSN*[,optional DPC_number]* - Determines whether all or a specified duplicate point code is accessible and, thereby, whether messages can be sent.	

Table A-1. MML Command Summary *(Page 28 of 29)*

MML Command	Description	Alternate/ Man Page Command
#SLD,x	Displays information about an application's load distribution (round-robin, least-utilized, or signaling link selection [SLS] distribution). Options for *x* include one of the following three #SLD command formats: • APPL,*appl_name* - Displays the SLS assignments for an application that registered using its name instead of its SSN (where *appl_name* is the name of the application). Use this format for applications that implement enhanced message distribution (for example, if an application is one of several applications that use the same SSN). For example, the command, **#SLD,APPL,DB12**, specifies a load control application named DB12. • SIO,*sio_number* - Displays the SLS assignments for an application that registered with a service information octet (SIO) (where *sio_number* is the SIO number). For example, the **#SLD,SIO,5** command specifies an application that registered with an SIO of 5. • SSN,*subsystem_number* - Displays the SLS assignments for an application that registered with a subsystem number (SSN) where *ssn_number* is the SSN of the application. For example, the **#SLD,SSN,254** command specifies an application that registered with an SSN of 254.	

Table A-1. MML Command Summary *(Page 29 of 29)*

MML Command	Description	Alternate/ Man Page Command
#STA,*x*,*n*	Displays the static tables. Options for *x* include: • cls - Displays combined link set tables. • cpc*[,optional SSN_number]* - Displays tables for all concerned point codes (CPCs) or for a specified SSN. • cr,index - Displays the cluster address table. • dt - Displays the discrimination table. • gtt - Displays the Global Title Translation entries. • lc*[,optional SSN_number]* - Displays all LC tables or those for a specified SSN. • lst - Displays link set tables. • mr,index - Displays the member address table. • ncpc*[,optional NETWORK_number]* - Displays all network concerned point codes (CPC) or those for a specified network. • nr - Displays the network address table. • rssn - Displays the remote SSN table. • st,*x* - Displays timer values. Options for *x* include: L2, L3, SCCP, Q707, MTP, PURGE, and ALL.	
sy	Invokes the sy utility.	
#sys	Displays the current settings of the shared memory system table.	
#UCOMM,x	Displays trace and status for UCOMM. Specify UCOMM number for *x*.	
Z	Displays structure sizes.	

Appendix B
SINAP/SS7 Environment Variables

This section lists and describes the environment variables for the SINAP/SS7 system and explains how to define them on the UNIX system.

Defining SINAP/SS7 Environment Variables

Environment variables define and activate operating characteristics on the SINAP node. The SINAP environment file `$SINAP_HOME/Bin/sinap_env.[sh|csh]` contains all SINAP/SS7 environment variables. Although most of the environment variables are disabled (commented out), some are activated (uncommented) during software configuration when you use the `/etc/config_sinap` script to configure the SINAP node.

NOTE

You must activate all environment variables to be enabled on a SINAP node before you start or restart the SINAP node.

Enabling Environment Variables

To activate an environment variable, do the following:

1. Stop SINAP.
2. Uncomment the relevant line(s) in the `sinap_env.[sh|csh]` file for that variable. The lines usually appear as follows:

   <variable-name>=*<value>*
   `export` *<variable-name>*

3. Log off and then log back into your SINAP account.
4. Use the `env` command to verify the new environment variable is set.
5. Start SINAP.

Note that for many of the environment variables, you do not need to assign values. The description associated with each environment variable in the SINAP environment file provide the valid setting(s), if required, for the variable.

Disabling Environment Variables

To deactivate an environment variable, do the following:

1. Stop SINAP.
2. Comment out the relevant line(s) in the `sinap_env.[sh|csh]` file for that variable. The lines usually appear as follows:

   <variable-name>=*<value>*
   `export` *<variable-name>*

3. Log off and then log back into your SINAP account.
4. Use the `env` command to verify the environment variable is no longer set.
5. Start SINAP.

The SINAP Environment File

This section provides the contents of the SINAP Environment file (`$SINAP_HOME/Bin/sinap_env_var.sh),` from which the file (`$SINAP_HOME/Bin/sinap_env.sh` is copied, for the Bourne shell.

sinap_env_var.sh (for Bourne Shell)

```
#Environment Variable                   Description
#================================================
#
#MANPATH /usr/share/man                 Enables the display of OS and
#                                       SINAP/MultiStack man pages. The
#                                       Installation script creates .login
#                                       and .profile files for sinap and
#                                       sysopr users and appends the SINAP
#                                       man page directory name to the
#                                       MANPATH variable for each user.
#
#                                       Variable is set through /etc/config_sinap

#PATH /usr/ucb                           Ensures you can display both OS and
#                                       SINAP/MultiStack man pages.

#SINAP_ALT_MEASUREMENT_INTERVAL         Specifies the reporting interval for
#                                       measurement reports (REPORT-MALL,
#                                       REPORT_MTP, REPORT_MSCCP, and
#                                       REPORT_MTCAP). Valid reporting
#                                       intervals are:
#                                       Value     Measurement Interval (Minutes)
#                                       -----     ------------------------------
#                                       0 or 30   30
#                                       1 or 15   15
#                                       2 or 5    5
#
#                                       If set to 0 or not defined, SINAP uses
```

```
#                                       a 30-minute measurement interval.
#SINAP_ALT_MEASUREMENT_INTERVAL=<value>
#export SINAP_ALT_MEASUREMENT_INTERVAL

#SINAP_CONSOLE_ALARM_LEVEL              Specifies the severity level of alarms that
#                                       SINAP/Multistack logs to the system error
#                                                 log file. Valid alarm values are:
#
#                                       Value     Description of Condition
#                                       -------   -------------------------------
#                                       CRITICAL  Causes severe service disruption
#                                                 and requires immediate attention
#                                                 (be sure to specify 'critical'
#                                                 in the Severity field of the
#                                                 message format as defined in the
#                                                 Emsg file).
#
#                                       MAJOR     Causes serious service disruption.
#                                                 Be sure to specify either
#                                                 'NONRECOVERABLE_L' or
#                                                 NONRECOVERABLE_LP - the problem
#                                                 cannot resolve itself and no
#                                                 acceptable alternative exists
#                                                 to address situation - in the
#                                                 Severity field of the message
#                                                 format as defined in Emsg file.
#
#                                       MINOR     Is not likely to cause a serious
#                                                 service disruption (be sure to
#                                                 specify INFP_L or INFO_LP in the
#                                                 severity field of the message
#                                                 format as defined in Emsg file).
#
#
#                                       NOTICE    Message is provided for
#                                                 informational purposes only.
#
#SINAP_CONSOLE_ALARM_LEVEL=<value>
#export SINAP_CONSOLE_ALARM_LEVEL

#SINAP_HEALTH_INTERVAL                  Specifies the time interval (in seconds)
#                                       for sending health-check requests
#                                       (default is 60 sec.).
#SINAP_HEALTH_INTERVAL=<value>
#                                       Variable is set through /etc/config_sinap

#SINAP_HEALTH_TIMEOUT                   Defines the interval (in seconds) within
#                                       which a process must respond to a
#                           health-check request. If two consecutive#
health-check requests fail, the process
#                                       is considered to have failed and trouble
#                                       management takes control (default is
#                                       60 sec.)
#SINAP_HEALTH_TIMEOUT=<value>
#                                       Variable is set through /etc/config_sinap

#SINAP_HOME                             Specifies the path name of your SINAP
#                                       account. For example, /home/sinap.
```

```
#SINAP_HOME=<value>
#                                   Variable is set through /etc/config_sinap

#SINAP_MML_LEVEL                    Defines the privilege level for
#                                   executing MML commands through the
#                                   Terminal Handler. Valid range is 0-255
#                                   (default is 5).
#SINAP_MML_LEVEL=<value>
#export SINAP_MML_LEVEL

#SINAP_HOME/Bin                     Adds the directory $SINAP_HOME/Bin to
#                                   your PATH environment variable.
#
#                                   Variable is set through /etc/config_sinap

#SINAP_LOG_SIZE                     Defines the size (in bytes) of the
#                                   Software Notebook and Alarm Log. The
#                                   MML report commands such as REPORT-ALARM
#                                   and REPORT-MALL) use this variable
#                                   (default value is 1000000 bytes).
#SINAP_LOG_SIZE=<value>
#                                   Variable is set through /etc/config_sinap

#SINAP_MDF                          Defines the path name of the Terminal
#                                   Handler's Menu Definition file.
#SINAP_MDF=<value>
#export SINAP_MDF

#SINAP_MODULE                       Specifies the module name assigned to a
#                                   SINAP/MultiStack system (default is M1).
#                                   You define this variable when you
#                                   configure the SINAP node using the
#                                   etc/config_sinap script.
#SINAP_MODULE=<name>
#                                   Variable is set through /etc/config_sinap

#SINAP_NODE                         Specifies the name assigned to a SINAP
#                                   node within a module (default is N1).
#                                   You define this variable when you
#                                   configure the SINAP node using the
#                                   etc/config_sinap script.
#SINAP_NODE=<name>
#                                   Variable is set through /etc/config_sinap

#SINAP_VARIANT                      Defines the network variant of the
#                                   SINAP node installed on the system.
#                                   Valid values are CCITT, ANSI, TTC,
#                                   NTT, and China. You define this
#                                   variable on the SINAP node using
#                                   the etc/config_sinap script.
#SINAP_VARIANT=<value>
#                                   Variable is set through /etc/config_sinap

#ENHANCED MESSAGE DISTRIBUTION
#============================

#DISCARDS_PER_ALARM                 Specifies the number of MSUs the
#                                   SINAP node discards before generating
```

```
#                                       an alarm.
#DISCARDS_PER_ALARM=<value>
#export DISCARDS_PER_ALARM

#UDTS_NO_OPC                            Specifies whether or not the SINAP
#                                       node generates a unitdata service
#                                       (UDTS) message when the MSU's
#                                       originating point code(OPC)is not
#                                       valid for the specified subsystem
#                                       number (SSN).
#UDTS_NO_OPC=1
#export UDTS_NO_OPC

#NETWORK VARIANT FEATURES
#============================

#ANSI_SINAP_FOPC                        (ANSI only). Activates the fictitious
#                           originating point code(FOPC) feature.
#ANSI_SINAP_FOPC=YES
#export ANSI_SINAP_FOPC

#TTC_WITH_NSTATE                        (TTC, NTT only). Enables applications
#                                       to support concerned point codes
#                                       (CPCs), which are responsible for
#                                       sending user-in-service(UIS) messages
#                                       and user-out-of-service (UDS) messages.
#TTC_WITH_NSTATE=1
#export TTC_WITH_NSTATE


#CCITT_CONGESTION_OPTION                (CCITT and China only). Specifies one of
#                                       the following methods used to handle link
#                                       congestion:
#
#                                       INTERNATIONAL_1_CONGESTION
#                                       NAT_MUL_CONG_WITH_PRIO
#                                       NAT_MUL_CONG_WO_PRIO
#
#INTERNATIONAL_1_CONGESTION             Specifies the international signaling
#                                       network option (the default), which
#                                       provides a single congestion onset
#                                       threshold and abatement threshold.
#
#CCITT_CONGESTION_OPTION=INTERNATIONAL_1_CONGESTION
#export CCITT_CONGESTION_OPTION

#NAT_MUL_CONG_WITH_PRIO                Specifies the national signaling network
#                                      option, multiple signaling link congestion

#                                      levels with congestion priority. In the
#                                      CCITT and China network variants, this
#                                      option allows client applications to
#                                      set congestion priority based on multiple
#                                      congestion levels (0-3). This option uses
#                                      these thresholds:
#
#                                      Congestion Onset (CONON1, CONON2, CONON3)
#                                      Congestion Abatement (CONAB1, CONAB2,
```

```
#                                   CONAB3)
#                                   Congestion Discard (CONDIS1, CONDIS2,
#                                   CONDIS3)
#
#                                   NOTE: ANSI, TTC, and NTT network variants
#                                   automatically implements this national
#                                   signaling network option (no environment
#                                   variable is required).
#
#CCITT_CONGESTION_OPTION=NAT_MUL_CONG_WITH_PRIO
#export CCITT_CONGESTION_OPTION

#NAT_MUL_CONG_WO_PRIO               Defines the national signaling network
#                                   option, multiple signaling link congestion
#                                   levels without congestion priority. For the
#                                   CCITT and China network variants, this
#                                   option allows the SINAP node to define up
#                                   to four levels (0-3) of link congestion
#                                   and to set a link's congestion status
#                                   according to those levels. When you set
#                                   this option, you must also specify
#                                   values for all the following environment
#                                   variables:
#
#                                   CONGESTION_STATUS
#                                   CONGESTION_INITIAL_VALUE
#                                   CONGESTION_TX_TIMER
#                                   CONGESTION_TY_TIMER
#
#CCITT_CONGESTION_OPTION=NAT_MUL_CONG_WO_PRIO
#export CCITT_CONGESTION_OPTION

#CONGESTION_STATUS                   Specifies the level (0,1,2, or 3) of link
#                                   congestion supported. The value 2 specifies
#                                   that the SINAP node supports three levels
#                                   (0, 1, and 2) of link congestion. The value
#                                   3 specifies support for all four link
#                                   congestion levels (0, 1, 2, and 3).
#
#
#CONGESTION_STATUS=<value>
#export CONGESTION_STATUS

#CONGESTION_INITIAL_VALUE            Defines the link congestion threshold
#                                   used to determine the occurrence
#                                   of congestion on a link. The Valid range
#                                   is 1-3.
#CONGESTION_INITIAL_VALUE=<value>
#export CONGESTION_INITIAL_VALUE

#CONGESTION_TX_TIMER                 Defines the interval in seconds between
#                                   congestion onset measurements. valid range
#                          is 1-255 (default is 1). When this timer #
expires, the SINAP node counts the number
#                                   of MSUs on the link's SS7 driver queue.
#                                   If the number of MSUs exceeds the value
#                                   of the congestion onset threshold, the
#                                   SINAP node increments the link's
```

```
#                                       congestion status by 1.
#CONGESTION_TX_TIMER=<value>
#export CONGESTION_TX_TIMER

#CONGESTION_TY_TIMER                    Defines the interval in seconds between
#                                       congestion abatement measurements. The
#                                       Valid range is 1-255 (default is 1).
#CONGESTION_TY_TIMER=<value>
#export CONGESTION_TY_TIMER

#RST_CONFIG_INIT_PROHIBIT                (CCITT only). Allows a newly created and
#                                        configured route set to be initialized
#                                        in the PROHIBITED state instead of the
#                                        ALLOWED state. This feature applies only
#                                        if the new route set being created and
#                                        configured is for a remote signaling
#                                        point in the network and not for an
#                                        adjacent point code (a signaling point
#                                        that is connected directly through a
#                                        link set to the SINAP node). For an
#                                        adjacent signaling point (including
#                                        adjacent STPs), the route set status
#                                        should be ALLOWED.
#
#                                        The interaction of this feature with MTP
#                                        restart is as follows:
#
#                                        If the SINAP node is performing MTP restart,
#                                        the new route sets configured during the
#                                        restart are set to the ALLOWED state. The
#                                        Adjacent STPs are responsible for sending
#                                        the appropriate transfer prohibited (TFP),
#                                        transfer restricted (TFR), or transfer
#                                        allowed (TFA) messages to the SINAP node
#                                        during the restart procedure.
#
#                                        If you use send_cm or sysopr to create
#                                        and configure a new route set to a
#                                        nonadjacent signaling point when SINAP
#                                        is not performing MTP restart, the route
#                                        set state is set to PROHIBITED.
#
#                                        Also, the SINAP node sends signaling
#                                        route set test (RST) messages to the
#                                        adjacent STPs for prohibited destinations
#                                        for which the new route sets were
#                                        configured. The node sends the RST
#                                        messages every timer T10 seconds until
#                                        the STPs  send TFA messages  to the
#                                        SINAP node.

#RST_CONFIG_INIT_PROHIBIT=1
#export RST_CONFIG_INIT_PROHIBIT

#LPC_ROUTING                             (CCITT, ANSI, China only). Enables the
#                                        Distributed Logical Point Code (DLPC)
#                                        feature that supports distributed ISUP
#                                        applications on two SINAP nodes. The
```

```
#                                       two ISUP applications are Logical Point
#                                       Codes (LPCs) that appear to the SS7
#                                       network as a pair of signaling transfer
#                                       points (STPs). Note that use of this
#                                       feature requires significant application
#                                       modifications.
#LPC_ROUTING
#export LPC_ROUTING

#MTP ENHANCED FUNCTIONALITY
#============================

#MTP_ANSI88_RSR_RST                     (ANSI only). Enables transfer-restricted
#                                       (TFR) message handling on the SINAP node
#                                       based on 1988 ANSI standards for MTP.
#                                       The node responds immediately to TFR or
#                                       TFP messages by sending signaling route
#                                       set restricted or prohibited (RSR or RSP)
#                                       messages without waiting for the T10
#                                       timer to expire.
#MTP_ANSI88_RSR_RST=1
#export MTP_ANSI88_RSR_RST

#MTP_ANSI92_MANAGEMENT_INHIBIT         (ANSI only). Enables management inhibiting
#                                      based on the 1992 ANSI Standards for MTP.
#                                      This variable allows a SINAP node to repeat
#                                      a link-forced uninhibit(LFU) message once
#                                      when the far end does not respond to the
#                                      first LFU request message with a link
#                                      uninhibit acknowledgement (LUN) message.
#                                      If the SINAP node does not receive a LUN
#                                      message from the far end, it discontinues
#                                      the request, then sends a link uninhibit
#                                      acknowledgment (LUA) message and starts
#                                      traffic on previously inhibited links. If
#                                      you do not define this variable, the SINAP
#                                      node implements management inhibiting
#                                      based  on 1988 ANSI Standards for MTP.
#MTP_ANSI92_MANAGEMENT_INHIBIT=1
#export MTP_ANSI92_MANAGEMENT_INHIBIT

#MTP_ANSI92_RESTART                   (ANSI only). Enables the MTP restart
#                                     feature which provides an orderly process
#                                     for activating the SINAP node's links and
#                                     routes when you start or restart the node.
#                                     This features is based on the 1992 ANSI
#                                     standards for MTP.
#MTP_ANSI92_RESTART=1
#export MTP_ANSI92_RESTART

#MTP_ANSI92_TCCO                     (ANSI only). Enables MTP timed-controlled
#                                    changeover (TCCO) functionality when the
#                                    SINAP node receives a processor outage
#                                    indication on a link at the remote end.
#                                    This feature is based on the 1992 ANSI
#                                    standards for MTP.
#
#MTP_ANSI92_TCCO=1
```

```
#export MTP_ANSI92_TCCO

#MTP_ANSI92_TCD                     (ANSI only). Enables use of the MTP
#                                   time-controlled diversion(TCD) feature,
#                                   based on the 1992 ANSI standards for MTP.
#                                   TCD is applied when the signaling point
#                                   made available at the far end of the link
#                                   is currently inaccessible from the signaling
#                                   point initiating the changeback order. If
#                                   you do not define this variable, the SINAP
#                                   node implements TCD functionality based on
#                                   the 1990 ANSI standards for MTP.
#MTP_ANSI92_TCD=1
#export MTP_ANSI92_TCD

#MTP_LINK_DOWN_AFTER_LPO_ENDS       Causes MTP processing to bring a link
#                                   down after a local processor outage(LPO).
#                                   You  must use normal link-level recovery
#                                   procedures to bring the link back into
#                                   service.
#MTP_LINK_DOWN_AFTER_LPO_ENDS=1
#export MTP_LINK_DOWN_AFTER_LPO_ENDS

#MTP_LINKSET_MEASUREMENT            Enables the REPORT-MMTP and REPORT-MALL
#                                   reports to display the number of octets
#                                   and MSUs transmitted and received per
#                                   second on all links configured for a
#                                   link set in 30-minute blocks during the
#                                   reporting interval specified in the MML
#                                   command. The reports also include the
#                                   peak octet and MSU values within the
#                                   30-minute interval for the reporting
#                                   period (day, week, month).
#MTP_LINKSET_MEASUREMENT=YES
#export MTP_LINKSET_MEASUREMENT

#MTP_RCT_LOAD_SHARING_SLS           (ANSI only).Enables the generation of
#                                   random signaling link selections(SLSs)
#                                   for outbound route set congestion test
#                                   (RCT) messages. If you do not define
#                                   this variable, the SINAP node implements
#                                   processing based on the 1988 ANSI standards
#                                   for MTP. Note that always sending the RCT
#                                   message on the same link within the same
#                                   link set on which the TFC was received
#                                   always results in the RCT message testing
#                                   the same network path, which may or may not
#                                   be congested.
#MTP_RCT_LOAD_SHARING_SLS=YES
#export MTP_RCT_LOAD_SHARING_SLS

#MTP_SLS4_LOAD_SHARE                (CCITT,China). Ensures message sequencing
#                                   by routing messages solely on the
#                                   signaling link selection (SLS) and
#                                   destination point code (DPC). For example,
#                                   if a telephone user part (TUP) user employs
#                                   the circuit identification code (CIC) for
#                                   the SLS value for all messages pertaining
```

```
#                                to that circuit, all messages go out over
#                                the same link to a particular DPC, even
#                                when loadsharing over two link sets. This
#                                ensures all messages remain in sequence.
#                                However, when loadsharing over link sets,
#                                you can only use 8 links in each link set.
#                                For multiple DPCs and more than 8 links per
#                                link set, you might achieve an even load
#                                distribution, but there is no guarantee.
#
#                                If you do not define this variable, the
#                                SINAP node supports 16 links per link set,
#                                with a random choice of link sets when
#                                loadsharing. This ensures even distribution,
#                                but not message sequencing. Note that
#                                enabling this option has no affect on TCAP,
#                               SCCP, or ISUP users.
#MTP_SLS4_LOAD_SHARE=1
#export MTP_SLS4_LOAD_SHARE

#MTP_USER_FLOW_CTL              Sends a user part unavailable(UPU)
#                               message to the originating user part
#                               when it receives an incoming message
#                               that it cannot deliver. This feature
#                               is based on the 1993 edition of the
#                               1993 ITU-T recommendations for MTP. If
#                               you do not define this variable, the
#                               SINAP node implements user flow control
#                               procedures based on the 1988 ITU-T
#                               recommendations for MTP. For the China
#                               network variant, you must define this
#                               variable.
#MTP_USER_FLOW_CTL=1
#export MTP_USER_FLOW_CTL

#MTP_WHITE_BOOK_RESTART         Enables the MTP restart feature that
#                               provides the orderly process for
#                               activating a node's links and routes
#                               when you start the SINAP node. This
#                               feature is based on the 1993 ITU-T
#                               recommendations for MTP. For the China
#                               network variant, you must define this
#                               variable.
#MTP_WHITE_BOOK_RESTART=1
#export MTP_WHITE_BOOK_RESTART

#MTP_WHITE_BOOK_SLC             Allows the SINAP node to accept certain
#                               types of signaling network management
#                               (SNM) messages that use a signaling link
#                               code(SLC) value other than 0. This feature
#                               is based on the 1993 ITU-T recommendations
#                               for MTP. For CCITT and China network
#                               variants, if you do not define this
#                               variable, the SINAP node implements SLS
#                               procedures based on the 1988 ITU-T
#                               recommendations for MTP. For the ANSI
#                               network variant, this feature is
#                               automatically activated. There is no
```

```
#                                    need to define this variable.
#MTP_WHITE_BOOK_SLC=1
#export MTP_WHITE_BOOK_SLC

#MTP_WHITE_BOOK_TCCO                 (CCITT and China). Enables use of the
#                                    MTP timed-controlled changeover(TCCO)
#                                    functionality when a processor outage
#                                    indication is received on a link at the
#                                    remote end. This feature is based on the
#                                    1988 ITU-T recommendations for MTP. If
#                                    you do not define this variable, the
#                                    SINAP node implements TCCO procedures
#                                    based on the 1988 ITU-T recommendations.
#MTP_WHITE_BOOK_TCCO=1
#export MTP_WHITE_BOOK_TCCO

#MTP_WHITE_BOOK_TFR                 (CCITT only). Implements the transfer
#                                   restricted (national network) message
#                                   handling option that is based on 1993
#                                   ITU-T Recommendations for MTP.
#MTP_WHITE_BOOK_TFR=1
#export MTP_WHITE_BOOK_TFR

#SLTM_NOT_REQUIRED                  When this environment variable is
#                                   defined, the SINAP node does not
#                                   initiate a signaling link test message
#                                   (SLTM) or for the TTC network variant,
#                           signaling route test message (SRTM).
#SLTM_NOT_REQUIRED=1
#export SLTM_NOT_REQUIRED

#SLTM_OPC_CHECK                     Checks the originating point code(OPC)
#                                   in the MTP routing label of the
#                                   received signaling link test
#                                   message(SLTM) to ensure it matches
#                                   the adjacent destination point code
#                                   (DPC) to which the link (indicated
#                                   by the signaling link code (SLC) of the
#                                   SLTM is connected.
#SLTM_OPC_CHECK=1
#export SLTM_OPC_CHECK

#SLTM_WITH_NAT10_FOR_G500           (CCITT only). Allows the SINAP node to
#                                   send a Signaling Link Test Acknowledgment
#                                   (SLTA) message in response to a Signaling
#                                   Link Test(SLT) message ONLY when the SLT
#                                   message contains the value NAT10 in the
#                                   subservice field(SSF).
#SLTM_WITH_NAT10_FOR_G500=1
#export SLTM_WITH_NAT10_FOR_G500


#LOOPBACK_DISPLAY                   (CCITT only). Enables the SINAP node to
#                                   detect when a remote link is in a
#                                   loopback mode. In this case, the SINAP
#                                   node sets a loopback diagnostic indicator
#                                   to display the loopback status on the
#                                   DISPLAY-LINK screen.
```

```
#LOOPBACK_DISPLAY=1
#export LOOPBACK_DISPLAY

#DISABLE_MTP_DISCRIMINATION        (CCITT only). Disables MTP Level 3 point
#                                  code discrimination and allows any MSU
#                                  received by the SINAP node (regardless
#                                  of destination point code(DPC) to be
#                                  routed to the appropriate application
#                                  process). The SINAP node discards all
#                                  SCCP management MTP management messages
#                                  with a DPC that is not equal to own point
#                                  code. The SINAP node handles MTP and SCCP
#                                  messages with a DPC equal to own point code
#                                  according to the 1993 ITU-T recommendations
#                                  for MTP and SCCP.
#DISABLE_MTP_DISCRIMINATION=1
#export DISABLE_MTP_DISCRIMINATION

#SCCP ENHANCED FUNCTIONALITY
#============================
#
#RESPONSE_WITHOUT_SSN_CONFIGURED   Specifies that the SCCP outbound routing
#                                  control of the SINAP node should retain
#                                  all outbound messages that contain remote
#                                  SSNs (REMSSNs) that are not configured
#                                  on the SINAP node or REMSSNs that are
#                                  configured, but not in the ALLOWED
#                                  state. If you do not define this
#                                  variable, the SINAP node discards all
#                                  these outbound messages.
#RESPONSE_WITHOUT_SSN_CONFIGURED=1
#export RESPONSE_WITHOUT_SSN_CONFIGURED

#REMSSN_INIT_PROHIBIT              (CCITT only). Enables the SINAP node to
#                                  set the remote SSN status to PROHIBITED
#                                  when the remote SSN is created using
#                                  the CREATE-REMSSN MML command or when
#                                  SCCP management(SCMG) receives an
#                                  MTP-RESUME primitive from MTP_L3RC
#                                  after a SINAP MTP restart occurs.
#REMSSN_INIT_PROHIBIT=1
#export REMSSN_INIT_PROHIBIT

#SCMG_SSP_SST_HANDLING_DISABLED    Disables handling of the SCCP management
#                                  messages subsystem prohibited(SSP) and
#                                  subsystem test (SST) messages.
#SCMG_SSP_SST_HANDLING_DISABLED=1
#export SCMG_SSP_SST_HANDLING_DISABLED

#CCITT_XUDT_SCMG                   (CCITT only). Allows a SINAP node to
#                                  receive and process an SCCP subsystem
#                                  test (SST) in an extended unitdata
#                                  (XUDT) message. If the SSN specified in
#                                  the XUDT message is in the ALLOWED state,
#                                  the SINAP node sends a subsystem allowed
#                                  (SSA) message to the original calling
#                                  calling address. If you do not define this
#                                  variable, the SINAP node discards any XUDT
```

```
#                                  SCCP management (SCMG) messages it receives
#                                  because SCMG messages are normally handled
#                                  only by unitdata (UDT) messages.
#CCITT_XUDT_SCMG=1
#export CCITT_XUDT_SCMG

#TCAP ENHANCED FUNCTIONALITY
#============================
#
#BYPASS_SINAP_GLOBAL_TITLE_TRANSLATION  Implements global title(GT) addressing
#                                       capabilities, instead of global title
#                                       translation (GTT). GT addressing
#                                       capabilities allow the SINAP node to
#                                       pass GT messages without actually
#                                       translating the GT.
#
#BYPASS_SINAP_GLOBAL_TITLE_TRANSLATION=1
#export BYPASS_SINAP_GLOBAL_TITLE_TRANSLATION

#GLOBAL_TITLE_SSN_NO_CHECK              This variable bypasses the availability
#                                       check of the remote SSN in the SCCP
#                                       called party address field of the
#                                       outbound message and enables a SINAP
#                                       node to send messages to an STP with the
#                                       outbound MSU's SCCP called party address
#                                       (including SSN, global title, and routing
#                                       indicator set to "route on global title")
#                                       if the SSN is unknown (SSN=0).
#GLOBAL_TITLE_SSN_NO_CHECK=1
#export GLOBAL_TITLE_SSN_NO_CHECK

#GTT_BYPASS_NOAI_CHECK                  Expands the value range for the nature
#                                       of address(NOAI) indicator field in
#                                       global title entries. Valid values for
#                                       NOAI can be within the range 1-127.
#                                       If you do not define the variable,
#                                       the value specified for NOAI can only
#                                       be in the range of 1-4.
#GTT_BYPASS_NOAI_CHECK=1
#export GTT_BYPASS_NOAI_CHECK

#GTT_WITH_BACKUP_DPC_SSN                Enables a SINAP node to route MSUs
#                                       containing global title information
#                                       to an alternate SCCP at a remote node
#                                       if the primary SCCP is unavailable. If
#                                       both primary and secondary SCCPs are
#                                       unavailable, the SINAP node returns a
#                                       NOTICE error message.
#GTT_WITH_BACKUP_DPC_SSN=1
#export GTT_WITH_BACKUP_DPC_SSN

#HEX_GLOBAL_TITLE                      Enables the use of hexadecimal values
#                                      (A-F) in the global title string. If
#                                      you do not define this variable, you
#                                      can only use decimal numbers (0-9) in
#                                      global title strings.
#
#HEX_GLOBAL_TITLE=1
```

```
#export HEX_GLOBAL_TITLE

#PARTIAL_GTT                            Enables the use of partial
#                                       Global Title Translation by setting a
#                                       maximum and minimum number of digits.
#                                       If you do not define this variable,
#                                       full GTT will take place.
#
#                                       When you set this option, you can also
#                                       specify values for the following
#                                       environment variables:
#
#                                       MAX_PGTT_DIGITS
#                                       MIN_PGTT_DIGITS
#
#PARTIAL_GTT
#export PARTIAL_GTT

#MAX_PGTT_DIGITS
#                                       Specifies the maximum number of partial
#                                       global title digits when PARTIAL_GTT is
#                                       set. Valid range is from 1 to
#                                       MAX_GLOBAL_TITLE (up to 28 digits). The
#                                       default is 6.
#export MAX_PGTT_DIGITS=<value>

#MIN_PGTT_DIGITS
#                                       Specifies the minimum number of partial
#                                       global title digits when PARTIAL_GTT is
#                                       set. If the digits in the address to be
#                                       translated is less than MIN_PGTT_DIGITS
#                                       then full GTT will take place. The
#                                       default is 3.
#export MIN_PGTT_DIGITS=<value>

#SINAP_XUDT_SEGMENT_SIZE                (CCITT, China only). Defines a size for
#                                       the message segments that make up an
#                                       extended unitdata (XUDT) message that
#                                       is smaller than the maximum size allowed
#                                       (the default) for the network variant
#                                       configured on the SINAP node.
#
#                                       Valid values  Network Variant
#                                       ------------  -------------------------
#                                       0 - 254       CCITT (default=254 bytes)
#                                       1 - 251       ANSI  (default=251 bytes)
#SINAP_XUDT_SEGMENT_SIZE=<value>
#export SINAP_XUDT_SEGMENT_SIZE


#TCAP_MAX_SIZE_ORIG_TID_ONLY            Defines the value of the origination
#                                       transaction ID which must be four bytes
#                                       long (the maximum allowable size).
#TCAP_MAX_SIZE_ORIG_TID_ONLY=1
#export TCAP_MAX_SIZE_ORIG_TID_ONLY

#TCRELAY                                CCITT only). Enables a SINAP node to
#                                       pass the hop counter value received in
```

```
#                                       an XUDT message to a TCAP application.
#TCRELAY=1
#export TCRELAY

#INTERPROCESS COMMUNICATIONS
#============================

#GUARANTEED_IPC                         Ensures that critical interprocess
#                                       communications(IPC) messages are
#                                       delivered to their destinations, even
#                                       during period of heavy system load.
#                                       Defining this variable changes the
#                                       retry_count parameter within the CASL
#                                       ca_put_msg() function as follows:
#
#                                       If retry_count value is 0, the SINAP
#                                       node considers the IPC message to be
#                                       noncritical and the node does not
#                                       generate a critical alarm if it cannot
#                                       deliver the message. Instead. the node
#                                       returns an error to the user with errno
#                                       typically set to EAGAIN.
#
#                                       If retry_count value is greater than 0,
#                                       the SINAP node considers the IPC message
#                                       to be critical and it makes every effort
#                                       to deliver the message, including
#                                       restarting the node if necessary. The
#                                       ca_put_msg() function call generates a
#                                       critical alarm and returns an error only
#                                       if the node cannot deliver the message
#                                       due to an error condition other than
#                                       EAGAIN (for example, the called process
#                                       no longer exists.
#GUARANTEED_IPC=1
#export GUARANTEED_IPC

#MML COMMAND ENHANCEMENTS
#========================
#SINAP_MML_ALT_NUMERIC_CONVERSION       Allows entry of MML values for command
#                                       arguments in decimal, octal, or
#                                       hexadecimal notation.
#
#                                       Notation    Command Argument Value
#                                       --------    --------------------------
#                                       Hexadecimal 0x or 0X, followed by value
#                                       Decimal     Value only(no prefix needed)
#                                       Octal       0, followed by the value
#SINAP_MML_ALT_NUMERIC_CONVERSION=1
#export SINAP_MML_ALT_NUMERIC_CONVERSION

#SINAP_MML_PRINT_RESPONSE               Enables the automatic printing feature
#                                       in MML commands that includes the PRINT
#                                       argument. If you specify a printer in the
#                                       PRINT argument of an MML command, the
#                                       specified printer automatically prints
#                                       the command and all its output. The
#                                       feature remains active until you exit
```

```
#                                       the login session. You can specify a
#                                       different printer within a login session
#                                       by changing  the printer defined in the
#                                PRINT argument of the MML command.
#SINAP_MML_PRINT_RESPONSE=1
#export SINAP_MML_PRINT_RESPONSE

#SINAP_MML_PRINT_TTYNAME                Allows the TTY name to be printed on the
#                                       printed output of a measurement-
#                                       reporting command. The TTY name
#                                       identifies the sysopr window from which
#                                       the MML command was issued.
#SINAP_MML_PRINT_TTYNAME=1
#export SINAP_MML_PRINT_TTYNAME

#CONNECTION-ORIENTED SERVICES
#============================
#SINAP_LRN_FREEZE_TIMEOUT               Defines the number of seconds (up to
#                                       1800) before an unused LRN is released
#                                       and can be assigned to another LRM
#                                       structure.
#SINAP_LRN_FREEZE_TIMEOUT=<value>
#export SINAP_LRN_FREEZE_TIMEOUT

#SINAP_TOTAL_LR_MEMS                    Defines the number of local reference
#                                       memory (LRM) structures specified (up
#                                       to 2000).
#SINAP_TOTAL_LR_MEMS=<value>
#export SINAP_TOTAL_LR_MEMS

#SINAP_TOTAL_LR_NUMS                    Allocates the total number (up to 5000)
#                                       of LRNs that can be assigned to LRM
#                                       structures.
#SINAP_TOTAL_LR_NUMS=<value>
#export SINAP_TOTAL_LR_NUMS

#SINAP_USER_LR_MEMS                     Defines the maximum number of connections
#                                       (up to 2000) each application can have
#                                       open at any given  time.
#SINAP_USER_LR_MEMS=<value>
#export SINAP_USER_LR_MEMS

#ISUP Services Feature
#=====================

#ISUP_FEATURE                           Activates a  standard-specific or
#                                       country-specific version of the ISUP
#                                       services feature on a SINAP node.
#
#                                       For the ANSI network variant, the only
#                                       valid ISUP version is
#                                       ANSI
#
#                                       For the CCITT network variant, valid
#                                       ISUP versions are:
#                                       ACIF_G500
#                                       BELGIUM
#                                       CCITT
```

```
#                                       FRANCE1
#                                       GERMANY
#                                       ITALY
#                                       ITU97
#                                       NETHERLANDS
#                                       Q767

#                                       MEXICO
#                                       SWEDEN
#                                       TAIWAN
#                                       SPAIN

#
#                                       For the China network variant, the
#                                       valid ISUP version is
#                                       CHINA

#                                       For the NTT network variant, valid ISUP
#                                       versions are:
#                                       NTT
#                                       NTT_IC
#ISUP_FEATURE=<VERSION>
#export ISUP_FEATURE

#ISUP_DBL_SEIZE_BITS                    (ANSI only). Enables you to set a value
#                                       in the range 0-3 for the Double Seizing
#                                       Control Indicator field of the Circuit
#                                       Group Characteristics Indicator parameter
#                                       in the Circuit Validation Response(CVR)
#                                       message. The SINAP node uses this value
#                                       for all configured circuits on the node.
#                                       If you do not define this variable or you
#                                       specify a value outside the valid range,
#                                       the SINAP node uses the default value
#                                       (0x00) for the Double Seizing Control
#                                       Indicator.
#ISUP_DBL_SEIZE_BITS=<value>
#export ISUP_DBL_SEIZE_BITS

#ISUP_REL_NO_ADD_ACC                    (ANSI only). Disables the Automatic
#                                       Congestion Control(ACC) parameter in a
#                                       Release (REL) message.
#ISUP_REL_NO_ADD_ACC=1
#export ISUP_REL_NO_ADD_ACC

#ISUP_CQR_TRANS_FOR_UCIC                (ANSI only). Causes the state for any
#                                       unequipped circuits reported in a Circuit
#                                       Query Response (CQR) message to be
#                                       transient. Also prevents the circuits from
#                                       going into the transient (maintenance)
#                                       state when certain outage conditions exist
#                                       (for example, when the ISUP application or
#                                       its process manager is not running). If
#                                       you do not define this variable, the SINAP
#                                       node reports the state "unequipped" in a
#                                       CQR message. This action applies to all
#                                       unequipped circuits, not just those
#                                       managed by the application.
```

```
#ISUP_CQR_TRANS_FOR_UCIC=1
#export ISUP_CQR_TRANS_FOR_UCIC

#ISUP_NO_UCIC_REPLIES                  (ANSI only). Prohibits the SINAP node from
#                                      sending Unequipped Circuit Identification
#                                      Code(UCIC) messages to remote nodes in
#                                      response to messages destined for
#                                      unequipped or unconfigured circuits. This
#                                      affects all unequipped circuits, not just
#                                      those managed by the application. Normally,
#                                      the SINAP node sends UCIC messages to
#                                      remote nodes for unequipped circuits.
#ISUP_NO_UCIC_REPLIES=1
#export ISUP_NO_UCIC_REPLIES

#ISUP_UNEQ_REL_ACK                     Allows REL on unregistered circuits to
#(ANSI only)                           be replied to with RLC to prevent
#                                      remote switch from triggering T5

#ISUP_UNEQ_REL_ACK=1
#export ISUP_UNEQ_REL_ACK

#ISUP_GENERIC_NAME                     (ANSI only). Enables use of the Generic
#                                      Name Parameter in the initial Address
#                                      Message (IAM).
#ISUP_GENERIC_NAME=1
#export ISUP_GENERIC_NAME

#ISUP_CGBA_PER_2CGB                    (ANSI only). Activates the feature (based
#                                      on ANSI 1992 standards) that returns a
#                                      circuit group block acknowledgement (CGBA)
#                                      message to the originator of the circuit
#                                      group block (CGB) message whenever two
#                                      CGB messages are received within a 5
#                                      second timer period.
#ISUP_CGBA_PER_2CGB=YES
#export ISUP_CGBA_PER_2CGB

#ISUP_RSC_BLO_PER_EXP                  (ANSI only). Enables the SINAP node to
#                                      return a blocking (BLO) message immediately
#                                      after returning the Reset Circuit(RSC)
#                                      message and handle the sending of BLO
#                                      messages based on timer T12 and T13
#                                      timeouts.
#ISUP_RSC_BLO_PER_EXP=1
#export ISUP_RSC_BLO_PER_EXP
#
#
#ALLOW_GR317_REV4_CAUSE_VALUES      (ANSI only) Allows cause values 25 and 26, which
#                        are supported in GR-317 Rev. 4, when a SINAP #
node is configured for ANSI 92.
#
#ALLOW_GR317_REV4_CAUSE_VALUES
#export ALLOW_GR317_REV4_CAUSE_VALUES

#FIRMWARE_HEARTBEAT=1                  Used to enable the U916 firmware heartbeat
#export FIRMWARE_HEARTBEAT             mechanism. To enable this, please uncomment
#                                      the following two lines.
```

```
#
#FIRMWARE_HEARTBEAT=1
#export FIRMWARE_HEARTBEAT

#DO_RPQDUMP=1                      Used to enable the generation of an rpqdump file
#export DO_RPQDUMP                 when the ss7dmn detects a board failure.  To
#                                  enable, please uncomment the following two lines.
#
#DO_RPQDUMP=1
#export DO_RPQDUMP

#CONG_EVENT_REPORT_OFF=1             This variable controls the congestion report for every
#export CONG_EVENT_REPORT_OFF        1 out of 8 outbound messages per Q.704 section 11.2.3.1
#                                    via IPC messages, M_message_for_congested_[link,route,
#                                    destination], from CASL to L3RC to notify MTP3 users
#                                    via I_MTP_STATUS primitives. Under heavy traffic with
#                                    congested link/linkset/routeset, these IPC messages
#                                    may prevent the other significant IPC messages from
#                                    processing. Setting this variable to 1 or a non-zero
#                                    value will disable this report.
#CONG_EVENT_REPORT_OFF=1
#export CONG_EVENT_REPORT_OFF

#SGS_SRST_ROUTING                  (CCITT only). Enable the MTP L3 inbound
#                                  processing of SRST messages, i.e. RCT, RST
#                                  and RSR, at L3DT which forwards them to IP
#                                  gateway process, e.g. IPMT or IPSG which
#                                  acts like STP. Also, the other feature is
#                                  to turn off the DPC validation after
#                                  inbound GTT, if this env var and the other
#                                  one, DISABLE_MTP_DISCRIMINATION, are both
#                                  defined.
#SGS_SRST_ROUTING=1
#export SGS_SRST_ROUTING
#
#OSP_UPDATE_ENABLED                (CCITT Only). Enable the ability to change
#                                  own point code, without SINAP stop/start
#                                  or activating/deactivating SINAP MMLs
#OSP_UPDATE_ENABLED=1
#export OSP_UPDATE_ENABLED
#
#ISUP_UPU_FEATURE                  This enables UPU message to be sent to
#                                  each remote point code, for any incoming
#                                  ISUP message, when ISUP application is
#                                  down.
#ISUP_UPU_FEATURE=1
#export ISUP_UPU_FEATURE
#
#SCCP_ITU96_IMPORTANCE_PARM        (CCITT Only). Enable the ability to handle
#                                  the 1996 ITU-T Q.71x optional Importance
#                                  parameter included in SCCP XUDT/XUDTS MSUs.
#SCCP_ITU96_IMPORTANCE_PARM=1
#export SCCP_ITU96_IMPORTANCE_PARM

#CONG_STATUS_CHANGE_LOGGING=1        This variable is to enable or disable SINAP alarm
#                                    logging for link/linkset/routeset(DPC) congestion
#                                    status change. Setting this variable to 1 or a non-zero
#                                    value will enable the logging of the congestion status
```

```
#                                   change in SINAP alarm log file. Otherwise this
#                                   feature will not be enabled.
#CONG_STATUS_CHANGE_LOGGING=1
#export CONG_STATUS_CHANGE_LOGGING
#
#SCCP_BYPASS_CLG_ADDR_CHECK        (CCITT Only). Enable the ability to bypass
#                                  SCCP calling party address validation for
#                                  inbound and outbound MSUs.
#SCCP_BYPASS_CLG_ADDR_CHECK=1
#export SCCP_BYPASS_CLG_ADDR_CHECK
```

Appendix C
CASL Error Messages

When a CASL function call is unsuccessful, the system returns an error message and an error value in the function's `errno` field. The message indicates the reason for the failure, and the number is the `errno` value, or error number. You will need to provide this error number if you are unable to resolve the problem and need to call the Stratus Customer Assistance Center (CAC) for help.

This appendix describes, in numeric order, the error messages that might be returned by CASL function calls. The following chart presents the range of `errno` values assigned to each type of error message. These values are defined in the SINAP/SS7 `ca_error.h` include file.

***errno* Value**	**Subsystem**
1 – 256	UNIX or SS7 Driver
1000 – 1999	Node Management
2000 – 2999	CASL
3000 – 3999	TCAP
4000 – 4999	SCCP
5000 – 5999	MTP
6000 – 6999	BITE
7000 – 7999	Client application
8000 - 8999	ISUP

This appendix lists and describes each CASL error message you might encounter. The messages are listed by subsystem and range of errno values, or error numbers. This appendix is divided into the following sections:

- "UNIX and SS7 Driver Errors" describes the errors that the UNIX operating system and the SS7 drivers can return when an application issues a call to a CASL function.
- "Node Management Errors" describes the errors UNIX can return when an application issues a call to the node management process.

- "CASL Errors" describes the errors that the CASL can return, including general, registration, load control, and connection-oriented errors.
- "TCAP Errors" describes the errors that TCAP can return.
- "SCCP Errors" describes the errors that SCCP can return.
- "MTP Errors" describes the errors that MTP can return.
- "Built-In Test Environment (BITE) Errors" describes errors returned by the BITE system.
- "Application Errors" describes the errors that can be returned when an application attempts to register with the SINAP/SS7 system.

NOTE

See the *SINAP/SS7 ISDN User Part (ISUP) Guide* (R8053) for descriptions of the errors that can be returned by ISUP.

UNIX and SS7 Driver Errors

This section lists and describes the UNIX and SS7 driver error messages. These errors have a value range of 1 through 256 and 500, and are listed by error number.

`1 EPERM`

Indicates one of the following problems:

- The user ID of the sending process is not configured with the appropriate privileges, and its real or effective user ID does not match the real or saved user ID of the receiving process.
- The calling process is attempting to send the `SIGCONT` signal to a process that does not share the same session ID.

This error can be returned by the `ca_get_msg()` function.

2 ENOENT

Indicates that the calling process has called an unknown file or directory that cannot be located. This error can be returned by the functions `ca_get_msg()` and `ca_put_msg()`.

3 ESRCH

Indicates that no process or process group can be found corresponding to the specified process ID (PID). This error can be returned by the functions `ca_get_msg()` and `ca_put_msg()`.

4 EINTR

Indicates that the system received a signal that interrupted the read or the system call. This error can be returned by the following functions:

```
ca_flush_msu(), ca_put_msg(), ca_register(),
ca_get_msg(), ca_put_msu(), ca_withdraw(),
ca_get_msu()
```

5 EIO

Indicates that an input/output (I/O) error occurred during a read or write operation. This error can be returned by the following functions:

```
ca_flush_msu(), ca_get_msu(), ca_register(),
ca_get_key(), ca_put_event(), ca_withdraw(),
ca_get_msg(), ca_put_msu()
```

6 ENXIO

Indicates that the requested service cannot be performed on the specified subdevice because the device or address does not exist. This error can be returned by the following functions:

```
ca_flush_msu(), ca_get_msu(), ca_register(),
ca_get_key(), ca_put_event(), ca_withdraw(),
ca_get_msg()
```

7 E2BIG

Indicates that the list of arguments exceeds the maximum size allowed. This error can be returned by the CASL function `ca_get_msg()`.

8 ENOEXEC

Indicates that the command contained a format error when the system attempted to execute it.

9 EBADF

Indicates that an invalid open file number was specified. This error can be returned by the following functions.

```
ca_flush_msu(), ca_put_event(), ca_put_tc(),
ca_get_key(), ca_put_msg(), ca_register(),
ca_get_msg(), ca_put_msu(), ca_withdraw(),
ca_get_msu()
```

10 ECHILD

Indicates that the calling process is calling a child process that is not provisioned.

11 EAGAIN

Indicates that the queue is full and the system cannot execute any more commands. This error can be returned by the `ca_put_event()` and `ca_put_msg()` functions.

12 ENOMEM

Indicates that the memory resource is full and you should try again. This error can be returned by the following functions.

```
ca_flush_msu(), ca_put_msu(), ca_put_tc()
```

13 EACCES

Indicates that the calling process has been denied permission to execute the requested operation. This error can be returned by the functions `ca_get_msg()` and `ca_put_msg()`.

14 EFAULT

Indicates that the pointer to the specified message is outside of the address space allocated for the calling process. This error can be returned by the following functions.

```
ca_flush_msu(), ca_get_msu(), ca_put_msu(),
ca_get_key(), ca_put_event(), ca_register(),
ca_get_msg(), ca_put_msg(), ca_withdraw()
```

15 ENOTBLK

Indicates that the operation requires a blocking device to complete the command. This error can be returned by the `ca_register()` function.

16 EBUSY

Indicates that the mount device is busy and the system cannot start the operation.

17 EEXIST

Indicates that `O_CREAT` and `OEXCL` are set and the named file exists. This error can be returned by the `ca_register()` function.

18 EXDEV

Indicates the calling process requires a cross-device link to complete the command.

19 ENODEV
Indicates that the operation requires a device that does not exist in the system.

20 ENOTDIR
Indicates that a component of the specified path name is not a directory. This error can be returned by the `ca_register()` function.

21 EISDIR
Indicates that the named file is a directory and the `oflag` is `write` or `read/write`. This error can be returned by the `ca_register()` function.

22 EINVAL
Indicates one of the following problems:

- The specified message queue ID is invalid.
- The value of `msg_type` is less than `1`.
- The value of `msg_sz` is greater than `0` or it exceeds the system-imposed limit.

This error can be returned by the following functions:

```
ca_flush_msu(), ca_get_msu(), ca_put_msg(),
ca_get_key(), ca_put_event(), ca_withdraw(),
ca_get_msg()
```

23 ENFILE
Indicates that the system file table is full. This error can be returned by the `ca_register()` function.

24 EMFILE
`NOFILES` file descriptors are currently open, indicating that the system has too many files open. This error can be returned by the `ca_register()` function.

25 ENOTTY
Indicates that the specified `fides` is not associated with a device driver that accepts control functions. This error can be returned by the following functions.

```
ca_flush_msu(), ca_put_event(), ca_put_tc(),
ca_get_key(), ca_put_msg(), ca_register(),
ca_get_msg(), ca_put_msu(), ca_withdraw(),
ca_get_msu()
```

26 ETXTBSY
Indicates that the calling process called a text file that is already in use.

27 EFBIG

Indicates the specified file is too large to open.

28 ENOSPC
Indicates one of the following problems:

- O_CREAT and OEXCL are set and the file system is out of I-nodes.
- The device does not exist and O_CREAT is specified.

This error can be returned by the ca_register() function.

29 ESPIPE
Indicates the calling process requested a search that it is not authorized to perform.

30 EROFS
Indicates that the named file resides on a read-only file system and oflag is set to write or read/write. This error can be returned by the ca_register() function.

31 EMLINK
Indicates there are too many links between processes and/or files.

32 EPIPE
Indicates the system has a broken pipe, or unusable queue.

33 EDOM
Indicates the operation contains a math argument that is out of the domain of the called function.

34 ERANGE
Indicates the response to the command contains a math result that cannot be represented.

35 ENOMSG
Indicates that the queue does not contain a message of the desired type. This error can be returned by the ca_get_msg() function.

36 EIDRM
Indicates that the identifier has been removed from the message and the call cannot be processed.

37 ECHRNG
Indicates that the requested channel number is out of range for the operation.

38 EL2NSYNC
Indicates that Level 2 functions are not synchronized.

39 EL3HLT
Indicates that all functions in Level 3 have stopped functioning.

40 El3RST

Indicates that the functions in Level 3 have been reset.

41 ELNRNG
Indicates that the requested link number is out of range for the operation.

42 EUNATCH
Indicates that the protocol driver is not attached and the operation cannot proceed.

43 ENOCSI
Indicates that no CSI structure is available.

44 EL2HLT
Indicates that all functions in Level 2 have stopped functioning.

45 EDEADLK
Indicates that the system has encountered a deadlock condition.

46 ENOLCK
Indicates that no record locks are available for the operation.

The following messages are returned for a convergent error:

50 EBADE
Indicates the system attempted an invalid exchange of data.

51 EBADR
Indicates the system used an invalid request descriptor.

52 EXFULL
Indicates the exchange queue is full.

53 ENOANO
Indicates the anode is unavailable or not provisioned.

54 EBADRQC
Indicates the operation request contained an invalid request code.

55 EBADSLT
Indicates an invalid slot in the configuration.

56 EDEADLOCK
Indicates the operation encountered a file locking deadlock error.

57 EBFONT
Indicates the request contained an invalid file font format.

The following messages indicate UNIX stream errors:

60 ENOSTR
Indicates the requested device is not a stream.

61 ENODATA
Indicates that there are no MSUs in the batch buffer. This error can be returned by the `ca_get_msu()` function.

62 ETIME
Indicates the timer expired before the operation completed.

63 ENOSR
Indicates the request requires the use of out of streams resources.

64 ENONET
Indicates the machine you are using is not connected to the network that is running the SINAP/SS7 system. Request assistance from your systems administrator to connect to the network.

65 ENOPKG
Indicates the operation requires use of a software package that is not installed or accessible from your machine. Request assistance from your systems administrator.

66 EREMOTE
Indicates the requested object is at a remote location and inaccessible at this time.

67 ENOLINK
Indicates that the link to the specified remote system is no longer active. This error can be returned by the following functions.

```
ca_flush_msu(), ca_get_msu(), ca_register(),
ca_get_key(), ca_put_event(), ca_withdraw(),
ca_get_msg()
```

68 EADV
Indicates an error in the UNIX system's ability to advertise.

69 ESRMNT
Indicates an error in using the UNIX `srmount` function.

70 ECOMM
Indicates a communication error occurred when the command was sent.

71 EPROTO

Indicates a UNIX protocol error.

74 EMULTIHOP
Indicates the system attempted a multihop, but was unable to complete the operation.

77 EBADMSG
Indicates the system is trying unsuccessfully to read an unreadable message.

78 ENAMETOOLONG
Indicates the command contains a path name that is too long.

79 EOVERFLOW
Indicates the command contains a field or parameter value too large to be stored in the data type.

80 ENOTUNIQ
Indicates the requested log name is not unique to the system. Rename the log and reissue the command.

81 EBADFD
Indicates the f.d. operation requested is invalid.

82 EREMCHG
Indicates the remote address specified in the command has changed.

The following error messages indicate problems in the system's shared libraries:

83 ELIBACC
Indicates the system cannot access the requested shared library. Review the correct path and library names, then reissue the request.

84 ELIBBAD
Indicates the shared library you are attempting to access has been corrupted and is unusable.

85 ELIBSCN
Indicates the `.lib` section in the `a.out` file has been corrupted and is inaccessible.

86 ELIBMAX
Indicates you attempted to link more libraries that the maximum allowed.

87 ELIBEXEC
Indicates you are attempting to execute a shared library.

88 EILSEQ
Indicates the operation command contains an illegal byte sequence.

89 ENOSYS
Indicates you are trying to execute a file operation that is not supported by the system.

90 ELOOP
Indicates the system is performing a symbolic link loop.

91 ERESTART
Indicates a restartable system call.

92 ESTRPIPE
Indicates that when the input queue is configured for first in, first out (FIFO) MSU processing for load control, there can be no delays or gaps in stream processing.

93 ENOTEMPTY
Indicates an attempt to delete or reconfigure a directory that still contains data.

94 EUSERS
Indicates the system has too many users (for UFS).

The following messages indicate errors in the UNIX Berkeley Software Distribution (BSD) system's arguments:

95 ENOTSOCK
Indicates an attempt to perform a socket operation on a non-socket element.

96 EDESTADDRREQ
Indicates the operation requires a destination address before it can be completed.

97 EMSGSIZE
Indicates the message is too long to display, send, or receive.

98 EPROTOTYPE
Indicates the protocol being used is the wrong type for the socket selected.

99 ENOPROTOOPT
Indicates the required protocol is unavailable.

120 EPROTONOSUPPORT
Indicates the system does not support the protocol requested/required.

121 ESOCKTNOSUPPORT
Indicates the system does not support the socket type required or requested.

122 EOPNOTSUPP
Indicates the system does not support the requested operation for the specified socket.

123 EPFNOSUPPORT
Indicates the system does not support the protocol family requested/required.

`124 EAFNOSUPPORT`
Indicates the protocol family in use does not support the address family requested.

`125 EADDRINUSE`
Indicates the address requested is already in use.

`126 EADDRNOTAVAIL`
Indicates the system cannot assign the requested address.

The following messages indicate UNIX or SS7 driver operational errors:

`127 ENETDOWN`
Indicates the network is out of service.

`128 ENETUNREACH`
Indicates the network is running, but the system cannot access it.

`129 ENETRESET`
Indicates a loss of network connection because the network is being reset.

`130 ECONNABORTED`
Indicates a loss of connection due to a software problem.

`131 ECONNRESET`
Indicates the connection was reset by a peer.

`132 ENOBUFS`
Indicates the system has no available buffer space.

`133 EISCONN`
Indicates the socket is already connected to the system.

`134 ENNOTCONN`
Indicates the socket requested/required is not connected to the system.

The following error messages pertain to the Microsoft XENIX system:

`135 EUCLEAN`
Indicates that the structure needs cleaning.

`137 ENOTNAM`
Indicates XENIX cannot recognize the file type because it is not named according to XENIX file naming conventions. Review the conventions, rename the file, and reissue the command.

`138 ENAVAIL`
Indicates no XENIX semaphores are available for use.

`139 EISNAM`
Indicates the requested file is a named type file.

`140 EREMOTEIO`
Indicates a remote input/output error.

`141 EINIT`
This error number is reserved for future use.

`142 EREMDEV`
This error number is reserved for future use.

The following messages are additional UNIX or SS7 driver operational errors:

`143 ESHUTDOWN`
Indicates the system cannot send commands because the socket has shut down.

`144 ETOOMANYREFS`
Indicates the system received or used too many references and cannot splice any additional ones.

`145 ETIMEDOUT`
Indicates a timer expired before the operation was complete and the connection timed out.

`146 ECONNREFUSED`
Indicates the requested connection could not be made.

`147 EHOSTDOWN`
Indicates the host system is out of service.

`148 EHOSTUNREACH`
Indicates the system cannot find a route to the requested host system.

`149 EALREADY`
Indicates the system already processed the command and the operation is in progress.

`150 EINPROGRESS`
Indicates the system is now processing the command.

The following error message pertains to the SUN Network File System (NFS):

`151 ECANCELED`
Indicates an outdated NFS file handle.

The following messages indicate errors in the loadable UNIX modules. To resolve the problem, review the system configuration for accuracy or request assistance from the system administrator.

`152 ENOLOAD`
Indicates the system cannot load the required module.

`153 ERELOAD`
Indicates the system encountered a relocation error while loading the requested module.

`154 ENOMATCH`
Indicates the system could not find a symbol matching the specification.

`155 EINPROG`
See `150 EINPROGRESS`.

`156 EBADVER`
Indicates the system cannot match the version number for the requested element to the version numbers in the system.

`157 ECONFIG`
Indicates the system has used all configured kernel resources.

`158 ECANCELED`
Indicates the system canceled the async input/output request.

The following message indicates an error in the Pyramid AIO compatibility raw disk asynchronous input/output function:

`500 EIORESID`
Indicates the data block was not fully transferred before processing stopped.

Node Management Errors

This section lists the node management error messages, including client management and disk server errors. Node management error values are in the range of 1000 through 1999.

The following errors are client management errors:

`1000 NMCL_NO_FREE_ENTRY`
Indicates the interprocess communications (IPC) table is full, allowing no additional entries.

`1001 NMCL_EXCEEDED_MAX_NODES`
Indicates the system has already used the maximum available nodes.

`1002 NMCL_EXCEEDED_MAX_MODULES`
Indicates the system has already used the maximum number of modules available.

`1003 NMCL_EXCEEDED_MAX_APPL`
Indicates the system has already activated the maximum number of applications allowed.

`1004 NMCL_EXCEEDED_MAX_PROCESSES`
Indicates the system is already running the maximum number of processes that can be run at one time.

`1005 NMCL_EXCEEDED_MAX_INST`
Indicates that you have exceeded the maximum number of instanciations (16) allowed at one time.

`1006 NMCL_KEY_ERROR`
Indicates that the SINAP/SS7 system cannot locate one or more pre-registered processes.

`1007 NMCL_ALREADY_REGISTERED`
Indicates client management already exists.

`1008 NMCL_TERM_KEY_INVALID`
Indicates the command contains an invalid IPC key to terminate processing.

`1009 NMCL_INVALID_LOAD_DIST`
Indicates inconsistent load distribution has been defined for multiple instanciations.

`1010 NMCL_INVALID_APPL_NAME`
Indicates an attempt to register with a different application name for an existing registered subsystem number/service information octet (SSN/SIO) subsystem.

`1011 NMCL_DUP_CTRL_PROC`
Indicates an attempt to register a duplicate control process.

`1012 NMCL_NAME_IN_USE`
Indicates an attempt to register a control process with the same name as a currently-registered data process.

`1013 NMCL_INVALID_DATA_NAME`
Indicates an attempt to register a duplicate data process with a different process name.

`1014 NMCL_INVALID_BOUNDARY`
Indicates an attempt to register duplicate data processes at different boundaries. For example, one process registers at the SCCP boundary and the other process registers at the SCCPX boundary.

The following messages indicate disk server errors:

`1020 NMDS_INVALID_CMD`
Indicates an unrecognizable command was entered.

`1021 NMDS_CMD_IN_PROGRESS`
Indicates the system is still processing a previous command and must complete the processing before beginning a new operation.

`1022 NMDS_INDEXING_ERROR`
Indicates the system could not find the IPC key. This is an internal error of the disk server.

`1023 NMDS_SHM_NO_LOAD`
Indicates the requested shared memory cannot be loaded.

`1024 NMDS_SHM_NO_BACK`
Indicates the requested shared memory cannot be backed up.

`1025 NMDS_INVALID_SHM_KEY`
Indicates the shared memory key is invalid.

`1026 NMDS_SHM_LOAD_ERROR`
Indicates an error occurred while the shared memory was being loaded.

`1027 NMDS_INVALID_COUNT`
The read/write data count is invalid (that is, it is too big or zero).

CASL Errors

This section presents a numeric listing of CASL error messages, including registration, general, load control, and connection-oriented errors. CASL errors are in the range of 2000 through 2999.

The following messages indicate registration function errors:

`2000 CA_ERR_ALREADY_REG`
Indicates that the application process is already registered with the SINAP/SS7 system and is attempting to register again. This error can be returned by the function `ca_register()`.

`2001 CA_ERR_REG_SSN`
The application process called `ca_register()` with an invalid subsystem number (SSN) in the `register_req_t` structure's `sio_ssn` field. Valid values are in the range `2` to `255`.

`2002 CA_ERR_REG_SIO`
The application process called `ca_register()` with an invalid service information octet (SIO) in the `register_req_t` structure's `sio_ssn` field. Valid values are in the range `1` to `15`.

`2003 CA_ERR_REG_SIO_SSN_IND`
The application process called `ca_register()` with an invalid value in the `register_req_t` structure's `sio_ssn_ind` field. This error indicates that the `sio_ssn_ind` value is either invalid, or it does not agree with the value of the `sio_ssn` or the `ss7_input_boundary` field (which determines whether an SIO or SSN is required). Valid values for `sio_ssn_ind` are as follows: `1` indicates that the `sio_ssn` field contains an SIO; `2` indicates an SSN; `3` indicates that the application process implements enhanced message distribution, in which case the `sio_ssn` field is set to `0` and the `dist_cmd_t` structure is used to define the SSN(s) to be associated with the application process. (See "Enhanced Message Distribution" and ''Custom Application Distribution'' in Chapter 3 for more information.)

`2004 CA_ERR_REG_SS7_BOUND`
The application process called `ca_register()` with an invalid value in the `register_req_t` structure's `ss7_input_boundary` field. Valid values are as follows: `1` (MTP), `2` (SCCP), and `3` (TCAP).

`2005 CA_ERR_REG_TCCOUNT`
The application process called `ca_register()` with an invalid value in the `register_req_t` structure's `tc_count` field.

`2006 CA_ERR_REG_SS7_PRIM`
The application process called `ca_register()` with an invalid value in the `register_req_t` structure's `ss7_primitive` field. Valid values are as follows: `1` (control primitives), `2` (data primitives), and `3` (control and data primitives).

`2007 CA_ERR_REG_LOAD_DIST`
Indicates that the application called `ca_register()` with the `register_req_t` structure's `inbound_load_dist_type` field set to an invalid value. Valid values are as follows: `1` (round robin), `2` (least utilized), and `3` (SLS distribution). For example, since the SLS distribution feature is available only to applications that receive input at the SCCP boundary or that register with an SIO instead of an SSN, `ca_register()` would return this error if the application process is registering to receive input at the TCAP boundary (`ss7_input_boundary` set to `3`) and is also attempting to specify a load distribution type of SLS distribution (`inbound_load_dist_type` set to `3`).

`2008 CA_ERR_REG_BATCHCOUNT`
Indicates that the application process called `ca_register()` with an invalid value specified for the `register_req_t` structure's `batch_count` field.

`2009 CA_ERR_REG_MAX_INMSU`
Indicates that the application process called `ca_register()` and did not specify a value for the `register_req_t` structure's `max_msu_input_que` field.

`2010 CA_ERR_REG_MAX_OUTMSU`
Indicates that the application process called `ca_register()` and did not specify a value for the `register_req_t` structure's `max_msu_out_que` field.

`2011 CA_ERR_REG_MAX_HOLD`
Indicates that the application process called `ca_register()` and did not specify a value for the `register_req_t` structure's `max_msu_holding_que` field.

`2012 CA_ERR_REG_MAX_TIME`
Indicates that the application process called `ca_register()` and did not specify a value for the `register_req_t` structure's `max_time_on_holding_que` field.

`2013 CA_ERR_REG_FAILURE_OPT`
Indicates that the application process called `ca_register()` with an invalid value specified for the `register_req_t` structure's `failure_option` field, which defines the action the SINAP/SS7 system is to perform if the registering process fails. Valid values are as follows: `1` (no action), `2` (send IPC message), or `3` (execute a script file).

`2014 CA_ERR_REG_SCR_FILE`
The application process called `ca_register()` with the `register_req_t` structure's `failure_option` field set to `3`; however, the `script` field does not contain the name of a script file, which is required when `failure_option` is set to `3`. This error can be returned by the `ca_register()` function.

`2015 CA_ERR_REG_MON_FILE`
Indicates that the application process called `ca_register()` with the `register_req_t` structure's `fmon_ss7` and/or `fmon_ipc` field set to `1`, which indicates that BITE monitoring is to be performed; however, the structure's `mon_filename` field does not specify the path name of the log file to which the BITE monitor messages are to be written.

`2016 CA_ERR_REG_INTC_FILE`
Indicates that the application process called `ca_register()` with the `register_req_t` structure's `fintercept` field set to `1`, which indicates that scenario execution is to be performed; however, the structure's `intc_filename` field does not specify the path name of a valid scenario execution program.

`2017 CA_ERR_REG_NORESP`
The application attempted to register with the SINAP/SS7 system; however, there is no response from the client management process. This error can be returned by the `ca_register()` function.

`2018 CA_ERR_REG_SS7_PRIMITIVE`
The application process called `ca_register()` with an invalid value in the `register_req_t` structure's `ss7_primitive` field. (For example, this error will occur if the application process sets the `register_req_t` structure's `fss7` field to `1` (use SS7 services) and sets `ss7_primitive` to `1` (accept control primitives).)

`2019 CA_ERR_REG_BCNT_HIGH`
Indicates that the application process called `ca_register()` with an MSU batch count that is greater than the driver queue count. To correct the problem, the application process must call `ca_register()` and specify a value for the `register_req_t` structure's `batch_count` field that is less than the values defined by the `max_msu_input_que` and `max_msu_out_que` fields.

The following messages indicate CASL general errors:

`2020 CA_ERR_NODE`
Indicates that the application process called `ca_get_key()` with an invalid node name.

`2021 CA_ERR_MODULE`
Indicates that the application process called `ca_get_key()` with an invalid module name.

`2022 CA_ERR_APPL`
Indicates that the application process called `ca_get_key()` with an invalid application name.

`2023 CA_ERR_PROC`

Indicates that the application process called `ca_get_key()` with an invalid process name specified in the function's `*pproc` parameter.

`2024 CA_ERR_INST`

Indicates that the application process called `ca_get_key()` with an invalid instance; that is, the instance is out of the range of instances associated with the application process.

`2025 CA_ERR_DESTN_KEY`

Indicates that the calling application process specified an invalid destination IPC key; that is, the specified IPC key is not listed in the IPC table. This error can be returned by the following functions.

```
ca_check_key(), ca_enable_intc(), ca_put_msg(),
ca_dbg_display(), ca_enable_mon(), ca_put_msg_def(),
ca_dbg_dump(), ca_get_key(), ca_put_reply(),
ca_disable_intc(), ca_put_cmd(), ca_restart_timer(),
ca_disable_mon(), ca_put_event(), ca_terminate()
```

`2026 CA_ERR_IBLK_DATA`

Indicates the calling application process specified `I_Block` data with a length that exceeds the maximum allowed. This error can be returned by the following functions.

```
ca_put_cmd(), ca_put_msg(), ca_put_reply(),
ca_put_event(), ca_put_msg_def(), ca_terminate()
```

`2027 CA_ERR_NO_SS7_SVC`

Indicates that the application process is not registered for SS7 services but has called a related function. To correct the problem, the application must reregister with the SINAP/SS7 system, specifying that it wants to use SS7 services. To do this, the application process must call `ca_register()` with the `register_req_t` structure's `fss7` field set to `1`; in addition, the structure's `ss7_primitive` field must be set to `2` or `3`. This error can be returned by the following functions.

```
ca_flush_msu(), ca_get_tc(), ca_put_tc(),
ca_get_msu(), ca_put_msu()
```

`2028 CA_ERR_CMDS`

Indicates that the application process is attempting to send a command to another process, but the other process is not registered to receive IPC commands. This error can be returned by the function `ca_put_cmd()`.

`2029 CA_ERR_NO_INTERCEPT`
Indicates that the application process is not registered to use the intercept mode. To correct the problem, the application must reregister with the SINAP/SS7 system, specifying that it wants to use the intercept mode.

`2030 CA_ERR_NO_MONITOR`
Indicates that the application process is not registered to use the monitor mode. To correct the problem, the application must reregister with the SINAP/SS7 system, specifying that it wants to the monitor mode.

`2031 CA_ERR_TIMER_ID_MSG`
Indicates that a message called a timer ID that does not exist for the message.

`2032 CA_ERR_MSU_CALLS`
This error can be returned by any of the following functions: `ca_get_msu()`, `ca_put_msu()`, `ca_get_tc()`, or `ca_put_tc()`. When returned in response to a `ca_get_msu()` or `ca_put_msu()` function call, this error indicates that the calling application process is registered for control primitives only, or it is registered to receive input at the TCAP boundary. To call either of these functions directly, an application must be registered to receive data primitives and it must registered to receive input at the MTP or SCCP boundary.

When returned in response to a `ca_get_tc()` or `ca_put_tc()` function call, this error indicates that the application cannot receive or send TCAP components because it is registered for control primitives only; to send or receive TCAP components, the application process must be registered to receive data primitives. (TCAP components are stored in MSUs; therefore, `ca_get_tc()` must call the `ca_get_msu()` function to retrieve the MSU containing the TCAP component. Similarly, `ca_put_tc()` calls the `ca_put_msu()` function to deliver a TCAP component, within an MSU, to the SS7 network.)

`2033 CA_ERR_IBLK_MSGTYPE`
Indicates that the calling application process specified an invalid message type for the `I_Block`. This error can be returned by the following functions.

`ca_put_msg(), ca_put_msg_def(), ca_terminate()`

`2034 CA_ERR_IBLK_PTR`
Indicates that the CASL function `ca_get_msg()` was called and the function's `piblk` parameter contained an invalid pointer to an `I_Block`.

`2035 CA_ERR_ACCESS`
Indicates that the calling application process is not registered with the SINAP/SS7 system. The application process must call the `ca_register()` function before calling any other CASL functions. This error can be returned by the following functions.

`ca_disable_intc(),ca_get_msu(), ca_put_msu(),`

`ca_disable_mon()`, `ca_health_chk_req()`, `ca_put_reply()`, `ca_dbg_display()`, `ca_health_chk_resp()`,`ca_put_tc()`, `ca_dbg_dump()`, `ca_put_cmd()`, `ca_register()`, `ca_enable_intc()`, `ca_put_event()`, `ca_restart_timer()`, `ca_enable_mon()`, `ca_put_msg()`, `ca_terminate()`, `ca_flush_msu()`, `ca_put_msg_def()`, `ca_withdraw()`, `ca_get_msg()`

`2036 CA_ERR_IPC_KEY`
Indicates that the specified `I_Block` is missing the `orig_id` and/or `dest_id` fields, or these fields refer to an invalid origination and/or destination IPC key. This error can be returned by the functions `ca_health_chk_resp()` and `ca_swap_keys()`.

`2037 CA_ERR_NO_MSUS`
Indicates that the application process called `ca_get_msu()` or `ca_get_tc()`, but there are no incoming MSUs on the queue. (TCAP components are stored in MSUs. Therefore, `ca_get_tc()` calls the `ca_get_msu()` function to retrieve the SS7 MSU that contains the TCAP component.)

`2038 CA_ERR_MBLK_SZ`
Indicates that the data read from the driver is equal to the size of the `mblock`(s).

`2039 CA_ERR_DFM_OVERFLOW`
Indicates that the table for the called timer is full and can contain no additional entries.

`2040 CA_ERR_MSG_TRUNCATED`
Indicates that the application process called `ca_dbg_display()` and the function's `pstring` parameter points to an ASCII string whose length exceeds 255 bytes, which is the maximum allowed.

`2041 CA_ERR_INT_MML`
Indicates that the Built-In Test Environment (BITE) subsystem returned an error when one of the following functions was called:

`ca_enable_intc()`, `ca_enable_mon()`, `ca_register()`

`2042 CA_ERR_TIMESTAMP_MSGTYPE`
Indicates that the function `ca_get_msg()` was called and contained an unknown timestamp message type.

`2043 CA_ERR_INVALID_MAXSZ`
Indicates that the function `ca_get_msg()` was called and the message on the IPC queue is larger than the maximum message size defined by the function's `max_sz` parameter. This error can be returned by the function `ca_get_msg()`.

The following CASL messages are associated with the load control process:

2044 CA_ERR_LC_BAD_ABDELAY
The ca_setup_locon() function call contains an invalid value for the *abate_delay* parameter. The value of *abate_delay* must be a decimal number in the range 1 to 10000.

2045 CA_ERR_LC_BAD_COUNT
The ca_setup_locon() function call contains an invalid value for the *count* parameter. The value of *count* must be a decimal number in the range 1 to 10000.

2046 CA_ERR_LC_BAD_DELAY
The ca_setup_locon() function call contains an invalid value for the *delay* parameter. The value of *delay* must be a decimal number in the range 1 to 10000.

2047 CA_ERR_LC_BAD_INST
The load control function call contains an invalid value for the *instance* parameter. Valid values are: 0 (INST_ALL), -1 (INST_THIS), or a decimal number in the range 1 to 16. If the *ssn* parameter is 0 (SSN_ALL), the *instance* parameter must also be 0.

NOTE

The functions ca_inquire_locon() and ca_setup_locon() do not use *instance*..

2048 CA_ERR_LC_BAD_NOTIFY
The ca_setup_locon() function call contains an invalid value for the *notify* parameter. Valid values are: 0 (LC_NONOTIFY), 1 (LC_NOTIFY), and 2 (LC_NOCHANGE).

2049 CA_ERR_LC_BAD_SSN
The load control function call contains an invalid value for the *ssn* parameter. Valid values are: -1 (SSN_THIS), 0 (SSN_ALL), or a decimal number in the range 2 to 255.

NOTE

The functions, ca_inquire_locon() and ca_invoke_locon(), do not support the value SSN_ALL.

2050 CA_ERR_LC_BAD_THRESH
The function call contains an invalid value for the *threshold* parameter. The value must be a decimal number in the range 1 to 10000.

2051 CA_ERR_LC_BAD_TYPE
The function call contains an invalid value for the *type* parameter. Valid values are: 1 (LC_GROUP), 2 (LC_INDIV), and 0 (LC_DELETE).

2052 CA_ERR_LC_DIST_WRONG

The specified application is configured for load control individual-type operation; however, the application is registered for `LEAST_UTILIZED` load distribution, which is not allowed for individual-type operation. Either re-register the application for `ROUND_ROBIN` load distribution (by calling `ca_register()` and specifying the value `1` for the *`inbound_load_dist_type`* parameter); or, reconfigure the application for load control group-type operation (by calling `ca_setup_locon()` and specifying the value `LC_GROUP` for the *`type`* parameter).

`2053 CA_ERR_LC_INST_DISABLE`
The specified application instance is not enabled for load control. To correct the problem, call the function to enable the application instance for load control.

`2054 CA_ERR_LC_INST_NOTRUN`
The specified application instance is not running. To correct the problem, either activate the application instance or call the load control function again, specifying an active application instance.

`2955 CA_ERR_LC_INST_NOT_FORCE`
The function was called for an application instance; however, the function must be called first.

`2056 CA_ERR_LC_NOTRUN`
The specified application is not running. Either activate the application or call the load control function again, specifying an active application.

`2057 CA_ERR_LC_NOT_ENABLE`
The specified application is not enabled for load control. Call the function to enable the application for load control.

`2058 CA_ERR_LC_NOT_FORCE`
The function was called for an application; however, the function must be called first.

`2059 CA_ERR_LC_NOT_INDIV`
The load control function call specifies an application instance; however, the application is not configured for load control individual-type operation. Reconfigure the application for individual-type operation by calling the function and specifying the value `LC_INDIV` for the *`type`* parameter.

NOTE

The application must have registered with an *`inbound_load_dist_type`* parameter value of `1`.

`2060 CA_ERR_LC_NOT_REG_SSN`
The specified application is registered with a service information octet (SIO). To use load control, the application must be registered with an SSN. Call `ca_register()` and re-register the application with an SSN instead of an SIO.

`2061 CA_ERR_LC_NOT_SETUP`
The specified application has not been configured for load control. Call the function to configure the application for load control.

`2062 CA_ERR_LC_NOT_TCAP`
The specified application is registered to receive input at the MTP or SCCP boundary. To use load control, an application must be registered to receive input at the TCAP boundary.

`2063 CA_ERR_LC_THRESHOLD`
The calling process (or application) attempted to configure the specified application for load control with a threshold value that is greater than the inbound MSU count. Either re-register the application so that its inbound MSU count (*`max_msu_input_que`*) is greater than its load control threshold value; or, lower the application's load control threshold value (by calling and specifying a value for *`threshold`* that is less than the inbound MSU count).

These following error messages indicate additional CASL registration, load control, connection-oriented, and general errors:

`2064 CA_ERR_REG_DIST_WRONG`
The calling process (or application) has been configured for load control individual-type operation. However, the application registered with `LEAST_UTILIZED` load distribution, which is not allowed for individual-type operation. Either re-register the application for `ROUND_ROBIN` load distribution (call `ca_register()` and specify `1` for the `register_req_t` structure's *`inbound_load_dist_type`* field), or reconfigure the application for load control group-type operation (call and specify `LC_GROUP` for the *`type`* parameter).

`2065 CA_ERR_REG_NOT_TCAP`
The application process called `ca_register()` and attempted to register to receive input at the SCCP boundary; however, the application is currently configured for load control and must therefore receive input at the TCAP boundary. To correct the problem, you must either reconfigure the application so that it does not use load control, or reregister the application to receive input at the TCAP boundary (call `ca_register()` with the `register_req_t` structure's `ss7_input_boundary` field set to `3`).

`2066 CA_ERR_REG_THRESHOLD`
The application process has been configured for load control and is now attempting to register with the SINAP/SS7 system with a maximum input MSU count that is too low for the application's load control threshold. To correct the problem, you can reconfigure the application's load control operating characteristics and reduce the load control threshold; or, reregister the application with the SINAP/SS7 system, increasing the value of the `register_req_t` structure's `max_msu_input_que` field. This error can be returned by the `ca_register()` function.

`2067 CA_ERR_NO_HOME_ENV`
Indicates that the application process called `ca_register()` to register with the SINAP/SS7 system but the `SINAP_HOME` environment variable was not defined. To correct the problem, define the variable and assign as its value the path name of the directory in which the SINAP/SS7 software is installed (for example, `SINAP_HOME=/home/sinap`).

NOTE

You must define the `SINAP_HOME` environment variable at the UNIX command level before starting the SINAP/SS7 system.

`2068 CA_ERR_FTOK`
Indicates that the application process is attempting to register with the SINAP/SS7 system, but the `SINAP_HOME` environment variable specifies a nonexistent path name. This error can be returned by the function `ca_register()`.

`2069 CA_ERR_NULL_DIST`
Indicates that the application process called `ca_register()` but the SINAP/SS7 system could not process the call because the `ssn_opc_table` argument was set to `NULL`.

`2070 CA_ERR_NULL_APPL`
Indicates that the application process called `ca_register()` but the SINAP/SS7 system could not process the call because the `register_req_t` structure's application value was set to zero (`0`).

`2071 CA_ERR_DIST_CMD`
Indicates that the application process called `ca_register()` but the SINAP/SS7 system could not process the call because it contained a bad distribution command.

`2072 CA_ERR_NEG_COUNT`
Indicates that the application process called `ca_register()` but the SINAP/SS7 system could not process the call because the `register_req_t` structure's `sio_ssn_ind` field was set to a negative value.

`2073 CA_ERR_MAX_COUNT`
Indicates that the application process called `ca_register()` but the SINAP/SS7 system could not process the call because the `register_req_t` structure contained SSN and OPC values that were too high.

`2074 CA_ERR_NOT_REG_MULT`
Indicates that the application process called `ca_register()` but the SINAP/SS7 system could not process the call because the `register_req_t`'s application SSN set was not set to `REG_MULT`. (See the section "Registering with SINAP/SS7" in Chapter 3 for additional considerations.)

`2075 CA_ERR_APPL_UNEQUIP`
Indicates that the application process called `ca_register()` but the SINAP/SS7 system could not process the call because the application was unequipped or unprovisioned.

`2076 CA_ERR_SSN_UNEQUIP`
Indicates that the application process called `ca_register()` but the SINAP/SS7 system could not process the call because the `register_req_t` structure's `sio_ssn_ind` field was unequipped.

`2077 CA_ERR_NO_OPC`
Indicates that the application process called `ca_register()` but the SINAP/SS7 system could not process the call because no OPC or SSN was defined for the application.

`2078 CA_ERR_MULT_CONFLICT`

Indicates a conflict in the configuration of the application that is calling `ca_register()`.

`2079 CA_ERR_REG_MULT_SSN`

The application process called `ca_register()` with an SIO or SSN specified in the `register_req_t` structure's `sio_ssn` field; however, the application is configured for enhanced message distribution and must therefore set `sio_ssn` to `0`.

You can correct the problem in one of two ways. Register the application process so that it does not implement enhanced message distribution by setting the `register_req_t` structure's fields as follows: set `sio_ssn_ind` to `1` (SIO) or `2` (SSN) and provide an SIO or SSN in the `sio_ssn` field. To have the application implement enhanced message distribution, set `sio_ssn_ind` to `3` (`REG_MULT`) and `sio_ssn` to `0`. (The `dist_cmd_t` structure defines the SSN(s) to associate with the application process. See "Enhanced Message Distribution" and ''Implementing Enhanced Message Distribution'' in Chapter 3 for information about enhanced message distribution and implementation.)

`2080 CA_ERR_REG_MULT_BOUND`

The application process attempted to register with the SINAP/SS7 system to receive input at the MTP boundary; however, the application is configured for enhanced message distribution and must therefore register to receive input at the SCCP or TCAP boundary. This error can be returned by the `ca_register()` function.

To correct the problem, either reconfigure the application so that it does not implement enhanced message distribution (see ''Enhanced Message Distribution'' in Chapter 3); or, reregister the application to receive input at the SCCP or TCAP boundary. To do this, call `ca_register()` with the `register_req_t` structure's `ss7_input_boundary` field set to `2` or `3`; in addition, make sure that the structure's `sio_ssn_ind` field is set to `2` and the `sio_ssn` field specifies an SSN (in the range `2` to `255`).

`2081 CA_ERR_REG_INCONSIST`
Indicates that the application process is attempting to register with an SIO or SSN that differs from the application's other processes. (Processes that are part of the same application must each register with the same SIO or SSN.) To correct the problem, the application process must reregister with the SINAP/SS7 system using the same SIO/SSN as the other application processes. (The `register_req_t` structure's `sio_ssn` field defines this SIO/SSN.)

`2082 CA_ERR_REG_APPL_FULL`
Indicates that the application process called `ca_register()` but the SINAP/SS7 system could not process the call because there are 32 applications currently registered, which is the maximum that the SINAP/SS7 system supports.

2083 `CA_ERR_REG_CONFLICT`
Indicates that the application process called `ca_register()` with the `register_req_t` structure's `sio_ssn_ind` field set to `REG_MULT`; however, one or more of the structure's other fields is set incorrectly. An `sio_ssn_ind` value of `REG_MULT` indicates that enhanced message distribution is being implemented, which means that `sio_ssn` must be set to `0` and `ss7_input_boundary` must be set to `2` (SCCP) or `3` (TCAP). (See "Registering with SINAP/SS7" in Chapter 3 for additional considerations.)

`2084 CA_ERR_REG_MAX_INST`
Indicates that the application process called `ca_register()` but the SINAP/SS7 system could not process the call because there are currently 16 application instances registered, which is the maximum that the SINAP/SS7 system allows.

`2085 CA_ERR_REG_INST_USED`
Indicates that the application process called `ca_register()` but the SINAP/SS7 system could not process the call because the specified instance number is already in use.

`2086 CA_ERR_LC_APPL_FULL`
Indicates that the load control function has encountered a full application table that cannot contain additional entries.

`2087 CA_ERR_LC_NO_APPL`
Indicates that no application exists for which to implement load control.

`2088 CA_ERR_REG_NODE_BAD`
Indicates that the application process called `ca_register()` but the SINAP/SS7 system could not process the call because there was a conflict with the `SINAP_NODE` specification.

`2089 CA_ERR_REG_MODULE_BAD`
Indicates that the application process called `ca_register()` but the SINAP/SS7 system could not process the call because there was a conflict with the SINAP_MODULE specification.

`2090 CA_ERR_REG_NODE_CONFIG`
Indicates that the application process called `ca_register()` but the SINAP/SS7 system could not process the call because the SINAP node specified was not configured.

`2091 CA_ERR_REG_VARIANT`
Indicates that the application process called `ca_register()` but the SINAP/SS7 system could not process the call because the application process and the SINAP node were configured with different network variants.

The following CASL messages pertain to SCCP Class 2 and Class 3 functions:

`2092 CA_ERR_REG_SCCP23`
The application did not register for connection-oriented services. Therefore, it cannot call `ca_put_sc()` or `ca_get_sc()`, as appropriate.

`2093 CA_ERR_REG_NSCCP23`
The application did not register for connection-oriented services. Therefore, it cannot call `ca_put_sc()` or `ca_get_sc()`, as appropriate.

`2094 CA_ERR_REG_SIZE_NG`
Indicates the application process called `ca_register()` with a user data size that is greater than the maximum size allowed.

`2095 CA_ERR_REG_CONN_NG`
Indicates the application process called `ca_register()`, but cannot register because the maximum number of connections have already been used.

`2096 CA_ERR_PUTSC_NG`
Indicates the `msg_type` data in the `m_block_t.sccp_ctrl` structure of `ca_register()` is invalid.

`2097 CA_ERR_PUTSC_BAD`
Indicates the specified `sccp_ctrl` value of `ca_register()` is invalid.

`2098 CA_ERR_PUTSC_SIZE`
Indicates the size of the MSU user data in `ca_register()` is invalid.

`2099 CA_ERR_PUTSC_BUSY`
Indicates no more connection IDs are available in the `conn-id` structure of `ca_put_sc()`.

`2100 CA_ERR_GETSC_BUSY`

Indicates no more connection IDs are available in the `conn-id` structure of `ca_put_sc()`.

`2101 CA_ERR_PUTSC_CID`
Indicates the specified connection ID in the MSU is too large for the buffer.

`2102 CA-ERR_GETSC_SIZE`
Indicates the incoming MSU's user data is larger that the size of the memory buffer used for storing the data. Note that the function call returns the portion of the MSU user data that fits in the memory buffer and discards the rest of the user data.

`2103 CA_ERR_PUTSC_MSG`
Indicates the application process is not registered as a Class 3 connection control process. Therefore, it cannot handle large messages.

`2104 CA_ERR_PUTSC_CONN`
Indicates the connection ID has been lost. Therefore, the SINAP/SS7 system cannot send the MSU.

`2105 CA_ERR_MTP_RESTART`
Indicates that CASL has to discard pending MSUs during MTP restart.

`2106 CA_ERR_GETSC_NG`
Indicates an invalid message type.

`2107 CA_ERR_REG_REACOUNT`
Indicates that zero was specified for the value in the `reassembly_count` field of the registration parameter structure `register_reg_t` of `ca_get_sc()`.

`2108 CA_ERR_XMAX_SIZE`
Indicates that the size of the XUDT data parameter exceeds the max of 2048, or the number of segments exceeds 16 (the maximum number defined in the 1993 edition of the ITU-T (CCITT) Recommendations for SCCP). The number of segments is determined by dividing the data size by the segments size, which is specified in the `SINAP_XUDT_SEGMENT_SIZE` environment variable.

`2109 CA_ERR_REG_MAX_INPUT_QUEUE`
Indicates that the application attempted to register (`tcrecv`) with CASL and failed one of two checks:

- CASL can reject the registering process if there are more MSUs in the input count queue (`max_msu_input_queue`) than the maximum of 32000 allowed. A value greater than 32000 means the streams threshold was crossed. This returns the registration error message above.
- However, if the application passes the CASL check, an additional check is made in the driver to ensure the registration of the new process does not result in a value that crosses the collective streams threshold, `strthresh`, divided by 2. This can be a value in the range of 7000 through 32000. If this happens, the driver returns the registration with the above error message.

`2110 CA_ERR_INVALID_PUT_REQ`
Indicates that the `ss7_input_boundry` is invalid for XUDT message processing.

The following CASL messages are associated with the custom application distribution process:

`2111 CA_ERR_CUST_APPL_INQ`
Indicates that the reverse application name lookup for SSN/OPC `DIST_INQ` operation is disallowed.

`2112 CA_ERR_NULL_APPL_LKUP`
Indicates a `NULL appl_name_table`.

`2113 CA_ERR_DIST_ID`
Indicates an invalid `custom_id`.

`2114 CA_ERR_CS1INAP_SVCKEY_CNT`
Indicates an invalid `svc_key_count`.

`2115 CA_ERR_CS1INAP_MAX_ENTRYS`
Indicates that the maximum number of `ssn`, `opc`, or `svc_key` entries were exceeded.

`2116 CA_ERR_CS1INAP_SET FAILED`

Indicates that the `ssn`, `opc`, or `svc_key` table update failed and the application was deleted from the tables.

This error can occur when the internal limit (257) on the number of ServiceKeys supported for a given SSN/OPC criteria is exceeded. This limit is imposed on the number of ServiceKey entries from all applications that specify the same SSN/OPC criteria. The number of required ServiceKeys/applications should be analyzed to determine if this number can be reduced, or if the SSN/OPC criteria can be re-arranged to avoid the limit.

This error can also occur in the event of an internal inconsistency that can be recovered without corruption of the internal driver tables. In this case, a system error is logged (not SINAP log, UNIX system log - console message). This indicates a serious condition. In this case report this error, along with all appropriate debug information, to the Ascend Customer Assistance Center (CAC).

`2117 CA_ERR_CS1INAP_NO_ENTRY`

Indicates that no entry matching the criteria specified was found.

This error is specific to the `DIST_INQ` command. The error indicates that no application with the specified criteria is currently running. This could mean that such an application can be started if the `DIST_INQ` operation was used to screen if the given criteria were already in use. Otherwise, this indicates that the specified criteria are not correct for any running applications, and the correct criteria should be used.

`2118 CA_ERR_CS1INAP_INCONSISTENT`

Indicates that an inconsistency was found in an internal driver table concerning the `DIST_INQ` parameter.

This error only occurs if an internal driver table inconsistency is discovered. At a minimum, the SINAP node should be restarted, and possibly the system rebooted. This indicates a serious condition. In this case report this error, along with all appropriate debug information, to the Stratus Customer Assistance Center (CAC).

TCAP Errors

This section presents a numeric listing of TCAP error messages. TCAP error messages use the abbreviations in the following chart.

Abbreviation	Meaning
ADDR	address
COMP	component(s)
CORR	correlation
DEST	destination
DIAL_ID	dialogue identifier
GEN	generate
GT	greater than
ID	identifier
IND	indicator
INV	invalid
ISM	invoke state machine
LEN	length
MSG	message
ORIG	origination
QWP	query with permission
QWOP	query without permission
REG	register
REJ	reject
SVC	service
TEQ	timer expiry queue
TOT	total
TRANS	transaction
TSL	transaction sublayer
USF	user-supplied function

TCAP error messages number in the range of 3000 through 3999. Some messages and error numbers are common to all network variants (CCITT, TTC, NTT, China, and ANSI). Other error messages are the same, but have slightly different error numbers, depending on the variant.

This section lists the error messages in numeric order, grouping the error message for the ANSI network variant and those for the CCITT/TTC/ NTT/China network variant.

`3000 TC_ERR_INV_TCAP_REG_PARAMETERS`
Indicates that the calling process attempted to register at the TCAP boundary with invalid parameters.

`3001 TC_ERR_NOT_REG_AT_TCAP_BOUNDARY`
Indicates that the calling process is not registered to receive input at the TCAP boundary. To correct the problem, the process must call `ca_register()` with the `register_req_t` structure's `ss7_input_boundary` field set to the value `3`, which indicates `SS7_INPUT_BOUNDARY_TCAP`. This error can be returned by the following functions.

`ca_alloc_tc()`, `ca_get_tc()`, `ca_put_tc()`, `ca_dealloc_tc()`, or `ca_process_tc()`

`3002` (ANSI) `TC_ERR_TRANS_ID_NOT_ALLOCATED`
Indicates that the application process called `ca_get_trans_id()` and no transaction ID was allocated for the `trans_id field`. Correct the problem by specifying a valid value for the `trans_id` parameter and reissuing the call.

`3002` (CCITT/TTC/NTT/China) `TC_ERR_DIAL_ID_ALREADY_RELEASED`
Indicates that the specified dialogue ID is no longer assigned to a dialogue; the ID has already been released. This error can be returned by the `ca_get_tc()` or `ca_rel_dial_id()` function. When this error is returned by `ca_get_tc()`, check to make sure that the function's `pfunc` parameter is `0` or that it specifies a pointer to a valid user-supplied function.

When returned by `ca_rel_dial_id()`, this error does not indicate a problem. Instead, it indicates that the specified dialogue ID has already been released and no further action is necessary.

`3003` (ANSI) `TC_ERR_TCAP_OWN_TRANS_ID`
Indicates that the application process called `ca_get_tc()` but the transaction is not under application control. Correct the problem by issuing a `TC_RESPONSE` primitive to release the transaction; no further action is required. (As defined by ANSI T1.114.5 Recommendations, a pre-arranged end causes messages to be discarded rather than being sent to the SS7 network.)

`3003` (CCITT/TTC/NTT/China) `TC_ERR_TCAP_OWN_DIAL_ID`
Indicates that the application process called `ca_get_tc()` but the dialogue is not under application control. Correct the problem by issuing a `TC_END` primitive to release the dialogue; no further action is required. (As defined by ITU-T (CCITT) Q.775 Recommendations, a pre-arranged end causes messages to be discarded rather than being sent to the SS7 network.)

`3004` (ANSI) `TC_ERR_OUT_OF_TRANS_ID_A`
Indicates that the calling process has exhausted its supply of transaction IDs. To correct the problem, the process must reregister with the SINAP/SS7 system, increasing the value of the `register_req_t` structure's `max_trans_id` field. This error, which can be returned by the `ca_get_tc()` or `ca_get_trans_id()` function, causes the SINAP/SS7 system to deallocate the `T_Block` structure; the function call is not executed.

NOTE

The primitives `TC_CONV_W_PERM` and `TC_CONV_WO_PERM`, `TC_RESPONSE`, `TC_P_ABORT`, and `TC_U_ABORT` do not return this error.

`3004` (CCITT/TTC/NTT/China) `TC_ERR_OUT_OF_DIAL_ID`
Indicates that the calling process has exhausted its supply of dialogue IDs. To correct the problem, the process must reregister with the SINAP/SS7 system, increasing the value of the `register_req_t` structure's `max_trans_id` field. This error, which can be returned by the `ca_get_tc()` or `ca_get_dial_id()` function, causes the SINAP/SS7 system to deallocate the `T_Block` structure; the function call is not executed.

NOTE

The primitives `TC_CONTINUE`, `TC_END`, `TC_P_ABORT`, and `TC_U_ABORT` do not return this error.

`3005` (ANSI) `TC_ERR_INV_TRANS_ID`
Indicates that the application process called `ca_put_tc()` with an invalid value specified for the `t_block_t` structure's `trans_id` field. Correct the problem by reissuing the `ca_put_tc()` function call and specifying for `trans_id` one of the following values: the transaction ID returned by a call to `ca_get_dial_id()` or the transaction ID from the `t_block_t` structure returned by a `ca_get_tc()` function call.

`3005` (CCITT/TTC/NTT/China) `TC_ERR_INV_DIAL_ID`
Indicates that the application process called `ca_put_tc()` with an invalid value specified for the `t_block_t` structure's `dialogue_id` field. Correct the problem by reissuing the `ca_put_tc()` function call and specifying for `dialogue_id` one of the following values: the dialogue ID returned by a call to `ca_get_dial_id()` or the dialogue ID from the `t_block_t` structure returned by a `ca_get_tc()` function call.

`3006` (ANSI) `TC_ERR_TRANS_ID_NOT_ASSIGNED`
Indicates that the application process called `ca_get_tc()` or `ca_put_tc()` without first calling `ca_get_trans_id()`. Before calling either of these functions, the process must call `ca_get_trans_id()` to obtain a transaction ID for the transaction.

`3006` (CCITT/TTC/NTT/China) `TC_ERR_DIAL_ID_NOT_ASSIGNED`
Indicates that the application process called `ca_get_tc()` or `ca_put_tc()` without first calling `ca_get_trans_id()`. Before calling either of these functions, the process must call `ca_get_dial_id()` to obtain a dialogue ID for the dialogue.

`3007 TC_ERR_OUT_OF_ISM_ENTRIES`
Indicates a serious condition. Report this error, along with all appropriate debug information, to the Stratus Customer Assistance Center (CAC). This error, which can be returned by the `ca_get_tc()` function, causes an immediate return; the function call is not executed.

The following error messages with numbers 3008 through 3033 are specific to the ANSI network variant.

`3008` (ANSI) `TC_ERR_T_BLOCK_CAPACITY_EXHAUSTED_A`
Indicates that the calling process has exhausted its supply of `t_block_t` (`T_Block`) structures. To correct the problem, the process must reregister with the SINAP/SS7 system, increasing the value of the `register_req_t` structure's `tc_count` field. This error, which can be returned by the `ca_alloc_tc()` or `ca_get_tc()` function, causes the SINAP/SS7 system to deallocate the `T_Block` structure; the function call is not executed.

3009 (ANSI) TC_ERR_INV_T_BLOCK_INDEX_A
Indicates that the application process called ca_dealloc_tc() or ca_put_tc() with an invalid value specified for the function's tb_index parameter, which specifies an index for one of the application's T_Blocks. This error indicates that the specified index is for a T_Block that is not part of the array of T_Blocks that the SINAP/SS7 system created for the application. Correct the problem by reissuing the call, specifying a valid value for the function's tb_index parameter. Valid values are in the range 0 to the application's maximum T_Block index minus 1.

NOTES

1. The number of T_Blocks that the SINAP/SS7 system allocates for the application's T_Block array is defined by the tc_count field of the register_req_t structure, which the application initializes prior to calling the ca_register() function.
2. Each T_Block index is relative to the global variable PTB, which is defined in the SINAP/SS7 ca_glob.h include file.

3010 (ANSI) TC_ERR_T_BLOCK_NOT_ALLOCATED_A
Indicates that the application process called ca_dealloc_tc() or ca_put_tc() without first calling ca_alloc_tc(). Before calling either of these functions, the application must call ca_alloc_tc() to allocate a T_Block.

3011 (ANSI) TC_ERR_TCAP_OWN_T_BLOCK_A
Indicates that the application process is attempting to manipulate a T_Block that is currently under TCAP's control. This error can be returned by the ca_put_tc() or ca_dealloc_tc() function. If this error is returned by ca_put_tc() and the application must release the T_Block, the application should issue a TC_U_CANCEL primitive, specifying the appropriate transaction and invoke IDs.

When returned by ca_dealloc_tc(), this error does not indicate a problem. Instead, it indicates that the specified T_Block is already under TCAP control; therefore, no further action is required.

3012 (ANSI) TC_ERR_INV_PRIMITIVE_CODE_A
Indicates that the application process called ca_put_tc() with an invalid value specified for the t_block_t structure's primitive_code field. See the description of ca_put_tc() in Chapter 6 for a list of valid values, which are defined in the SINAP/SS7 system's tblock.h include file. Correct the problem by reissuing the call, specifying a valid value for primitive_code.

3013 (ANSI) TC_ERR_INV_PRIORITY
Indicates that the application process called ca_put_tc() with an invalid value specified for the t_block_t structure's priority field. See the description of

`ca_put_tc()` in Chapter 6 for a list of valid values.

`3014` (ANSI) `TC_ERR_INV_QUALITY_OF_SVC_A`
Indicates that the application process called `ca_put_tc()` with an invalid value specified for the `qlty_of_svc` field of the `tc_thp_t` structure. (This field specifies the SCCP protocol class, `0` or `1`, both of which support connectionless service only.) Correct the problem by reissuing the call, specifying one of the following values for `qlty_of_svc`.

`0` – protocol class 0 (unsequenced), no return on error
`1` – protocol class 1 (sequenced), no return on error
`0x80` – protocol class 0 (unsequenced), return on error
`0x81` – protocol class 1 (sequenced), return on error

`3015` (ANSI) `TC_ERR_INV_ORIG_TOT_ADDR_LEN_A`
Indicates that the application process called `ca_put_tc()` with an SCCP calling-party address that is too long. (The SCCP calling-party address is defined in the `orig_addr` field of the `tc_thp_t` structure.) Correct the problem by reissuing the call, making sure that `orig_addr` defines an address of the correct length.

NOTE

The variable `MAX_ADDR_LEN` specifies the maximum address length. It is defined in the SINAP/SS7 `tblock.h` include file.

`3016` (ANSI) `TC_ERR_INV_DEST_TOT_ADDR_LEN_A`
Indicates that the application process called `ca_put_tc()` with an SCCP called-party (or destination) address that is too long. (The SCCP called-party address is defined in the `tc_thp_t` structure.) Correct the problem by reissuing the call, making sure that `dest_addr` defines an address of the correct length.

NOTE

The variable `MAX_ADDR_LEN` specifies the maximum address length, which is defined in the SINAP/SS7 `tblock.h` include file.

3017 (ANSI) TC_ERR_INV_TRANS_END_TYPE
In an attempt to end the transaction, the application process called ca_put_tc() with an invalid value specified for the tc_thp_t structure's trans_end_type field. Valid values are 1 (pre-arranged end) or 2 (basic end). Correct the problem by reissuing the ca_put_tc() function call, specifying a valid value for trans_end_type.

3018 (ANSI) TC_ERR_INV_CLASS_OF_SVC_A
Indicates that the application process called ca_put_tc() with an invalid value specified for the class_of_svc field of the of the tc_thp_t structure.

3019 (ANSI) TC_ERR_INV_TIMER_VALUE_A
Indicates that the application process called ca_put_tc()with a value of 0 specified for the t_chp_t structure's timer_value field. Correct the problem by reissuing the call, specifying a value other than 0 for the timer_value field.

3020 (ANSI) TC_ERR_INV_INVOKE_ID
Indicates that the application process called ca_put_tc() with an invalid value specified for the invoke_id field of the of the tc_thp_t structure. Correct the problem by specifying a valid value for this field.

3021 (ANSI) TC_ERR_INV_CORR_ID
Indicates that the application process called ca_put_tc() with an invalid correlation ID specified for the corr_id of the tc_thp_t structure. Correct the problem by specifying a valid value for this field.

3022 (ANSI) TC_ERR_DUPLICATE_INVOKE_ID_A
Indicates that the application process called ca_put_tc() with an invoke ID that is currently being used by another dialogue/transaction. Each dialogue/transaction requires a unique invoke ID. Correct the problem by reissuing the call, specifying a unique invoke ID for the tc_chp_t structure's invoke_id field.

3023 (ANSI) TC_ERR_ISM_NOT_IDLE_A
Indicates a serious condition. Report this error, along with all appropriate debug information, to the Stratus Customer Assistance Center (CAC). This error, which can be returned by the ca_get_tc() function, causes an immediate return; the function call is not executed.

3024 (ANSI) TC_ERR_COMP_LEN_GT_DATA_SIZE_A
Indicates that the application process issued a call with an invalid component length for the global title data size. Global title formats are described in "Routing Capabilities" in Chapter 3.

`3025` (ANSI) `TC_ERR_TOT_LEN_GT_MSU_SIZE_A`
Indicates that the application process issued a call with an invalid MSU size specified for the global title. Global title formats are described in "Routing Capabilities" in Chapter 3.

`3026` (ANSI) `TC_ERR_GEN_REJ_COMP_A`
Indicates a general error in the specification of TCAP components. Review specifications and correct any that are not valid.

`3027` (ANSI) `TC_ERR_LEN_IND_VALUE_ZERO_A`
Indicates that the application process issued a call to `ca_put_tc()` with the indicator value length set to zero, which indicates that there is no information defined. Correct the problem by reissuing the call, specifying valid information for the data field.

`3028` (ANSI) `TC_ERR_UNI_MSG_NO_COMP_A`
The application process attempted to send information to another application by calling `ca_put_tc()` with the `t_block_t` structure's `primitive_code` field set to `TC_UNI`. However, the `tc_chp_t` structure's `data` field has a length of `0`, which indicates that there is no information defined. Correct the problem by reissuing the call, specifying valid information for the data field.

`3029` (ANSI) `TC_ERR_NO_MSU_BUILT_A`
Indicates a TCAP application process attempted to process MSUs, but there are no MSUs built.

`3030` (ANSI) `TC_ERR_INV_TSL_STATE_A`
Indicates a serious condition. Report this error, along with all appropriate debug information, to the Stratus Customer Assistance Center (CAC). This error, which can be returned by the `ca_get_tc()` function, causes an immediate return; the function call is not executed.

`3031` (ANSI) `TC_ERR_INV_TSL_EVENT_A`
Indicates a serious condition. Report this error, along with all appropriate debug information, to the Stratus Customer Assistance Center (CAC). This error, which can be returned by the `ca_get_tc()` function, causes an immediate return; the function call is not executed.

`3032` (ANSI) `TC_ERR_INV_ISM_STATE_A`
Indicates a serious condition. Report this error, along with all appropriate debug information, to the Stratus CAC. This error, which can be returned by the `ca_get_tc()` function, causes an immediate return; the function call is not executed.

`3033` (ANSI) `TC_ERR_INV_ISM_EVENT_A`
Indicates a serious condition. Report this error, along with all appropriate debug information, to the Stratus Customer Assistance Center (CAC). This error, which can be returned by the `ca_get_tc()` or `ca_put_tc()` function, causes an immediate return; the function call is not executed.

The following error messages with numbers 3008 through 3033 are specific to the CCITT, TTC, NTT, and China network variants.

NOTE

If you are running the CCITT network variant, the error messages contain an additional `_C` suffix at the end of the message, for example, `TC_ERR_ISM_NOT_IDLE_C`.

`3008` (CCITT/TTC/NTT/China) `TC_ERR_TRANS_ID_ALREADY_RELEASED`
Indicates that the specified transaction ID is no longer assigned to a transaction; the ID has already been released. This error can be returned by the `ca_get_tc()` or `ca_rel_dial_id()` function. When this error is returned by `ca_get_tc()`, check to make sure that the function's `pfunc` parameter is `0` or that it specifies a pointer to a valid user-supplied function.

When returned by `ca_rel_trans_id()`, this error does not indicate a problem. Instead, it indicates that the specified transaction ID has already been released; therefore, no further action is necessary.

`3009` (CCITT/TTC/NTT/China) `TC_ERR_OUT_OF_TRANS_ID`
Indicates that a user-supplied function has exhausted its supply of user IDs. This error can be returned by the `ca_get_tc()` function.

`3010` (CCITT/TTC/NTT/China) `TC_ERR_T_BLOCK_CAPACITY_EXHAUSTED`
Indicates that the calling process has exhausted its supply of `t_block_t` (`T_Block`) structures. To correct the problem, the process must reregister with the SINAP/SS7 system, increasing the value of the `register_req_t` structure's `tc_count` field. This error, which can be returned by the `ca_alloc_tc()` or `ca_get_tc()` function, causes the SINAP/SS7 system to deallocate the `T_Block` structure; the function call is not executed.

`3011` (CCITT/TTC/NTT/China) `TC_ERR_INV_T_BLOCK_INDEX`
Indicates that the application process called `ca_dealloc_tc()` or `ca_put_tc()` with an invalid value specified for the function's `tb_index` parameter, which specifies an index for one of the application's `T_Blocks`. This error indicates that the specified index is for a `T_Block` that is not part of the array of `T_Blocks` that the SINAP/SS7 system created for the application. Correct the problem by reissuing the call, specifying a valid value for the function's `tb_index` parameter. Valid values are in the range `0` to the application's maximum `T_Block` index minus `1`.

NOTES

1. The number of `T_Blocks` that the SINAP/SS7 system allocates for the application's `T_Block` array is defined by the `tc_count` field of the `register_req_t` structure, which the application initializes prior to calling the `ca_register()` function.
2. Each `T_Block` index is relative to the global variable `PTB`, which is defined in the SINAP/SS7 `ca_glob.h` include file.

`3012` (CCITT/TTC/NTT/China) `TC_ERR_T_BLOCK_NOT_ALLOCATED`
Indicates that the application process called `ca_dealloc_tc()` or `ca_put_tc()` without first calling `ca_alloc_tc()`. Before calling either of these functions, the application must call `ca_alloc_tc()` to allocate a `T_Block`.

`3013` (CCITT/TTC/NTT/China) `TC_ERR_TCAP_OWN_T_BLOCK`
Indicates that the application process is attempting to manipulate a `T_Block` that is currently under TCAP's control. This error can be returned by the `ca_put_tc()` or `ca_dealloc_tc()` function. If this error is returned by `ca_put_tc()` and the application must release the `T_Block`, the application should issue a `TC_U_CANCEL` primitive, specifying the appropriate transaction and invoke IDs.

When returned by `ca_dealloc_tc()`, this error does not indicate a problem. Instead, it indicates that the specified `T_Block` is already under TCAP control; therefore, no further action is required.

`3014` (CCITT/TTC/NTT/China) `TC_ERR_INV_PRIMITIVE_CODE`
Indicates that the application process called `ca_put_tc()` with an invalid value specified for the `t_block_t` structure's `primitive_code` field. See the description of `ca_put_tc()` in Chapter 6 for a list of valid values, which are defined in the SINAP/SS7 system's `tblock.h` include file. Correct the problem by reissuing the call, specifying a valid value for `primitive_code`.

`3015` (CCITT/TTC/NTT/China) `TC_ERR_INV_QUALITY_OF_SVC`
Indicates that the application process called `ca_put_tc()` with an invalid value specified for the `qlty_of_svc` field of the of the `tc_dhp_t` structure. (This field

specifies the SCCP protocol class, `0` or `1`, both of which support connectionless service only.) Correct the problem by reissuing the call, specifying one of the following values for `qlty_of_svc`.

`0` – protocol class 0 (unsequenced), no return on error
`1` – protocol class 1 (sequenced), no return on error
`0x80` – protocol class 0 (unsequenced), return on error
`0x81` – protocol class 1 (sequenced), return on error

`3016` (CCITT/TTC/NTT/China) `TC_ERR_INV_ORIG_TOT_ADDR_LEN`
Indicates that the application process called `ca_put_tc()` with an SCCP calling-party address that is too long. (The SCCP calling-party address is defined in the `orig_addr` field of the `tc_dhp_t` structure.) Correct the problem by reissuing the call, making sure that `orig_addr` defines an address of the correct length.

NOTE

The variable `MAX_ADDR_LEN` specifies the maximum address length. It is defined in the SINAP/SS7 system's `tblock.h` include file.

`3017` (CCITT/TTC/NTT/China) `TC_ERR_INV_DEST_TOT_ADDR_LEN`
Indicates that the application process called `ca_put_tc()` with an SCCP called-party (or destination) address that is too long. (The SCCP called-party address is defined in the `dest_addr` field of the `tc_dhp_t` structure.) Correct the problem by reissuing the call, making sure that `dest_addr` defines an address of the correct length.

NOTE

The variable `MAX_ADDR_LEN` specifies the maximum address length, which is defined in the SINAP/SS7 `tblock.h` include file.

`3018` (CCITT/TTC/NTT/China) `TC_ERR_INV_DIAL_END_TYPE`
In an attempt to end the dialogue, the application process called `ca_put_tc()` with an invalid value specified for the `tc_dhp_t` structure's `dialogue_end_type` field. Valid values are `1` (pre-arranged end) or `2` (basic end). Correct the problem by reissuing the `ca_put_tc()` function call, specifying a valid value for `dialogue_end_type`.

`3019` (CCITT/TTC/NTT/China) `TC_ERR_INV_CLASS_OF_SVC`
Indicates that the application process called `ca_put_tc()` with an invalid value specified for the `class_of_svc` field of the of the `tc_thp_t` structure.

`3020` (CCITT/TTC/NTT/China) `TC_ERR_INV_TIMER_VALUE`
Indicates that the application process called `ca_put_tc()` with a value of `0` specified for the `t_chp_t` structure's `timer_value` field. Correct the problem by reissuing the call, specifying a value other than `0` for the `timer_value` field.

`3021` (CCITT/TTC/NTT/China) `TC_ERR_DUPLICATE_INVOKE_ID`
Indicates that the application process called `ca_put_tc()` with an invoke ID that is currently being used by another dialogue/transaction. Each dialogue/transaction requires a unique invoke ID. Correct the problem by reissuing the call, specifying a unique invoke ID for the `tc_chp_t` structure's `invoke_id` field.

`3022` (CCITT/TTC/NTT/China) `TC_ERR_ISM_NOT_IDLE`
Indicates a serious condition. Report this error, along with all appropriate debug information, to the Stratus Customer Assistance Center (CAC). This error, which can be returned by the `ca_get_tc()` function, causes an immediate return; the function call is not executed.

`3023` (CCITT/TTC/NTT/China) `TC_ERR_COMP_LEN_GT_DATA_SIZE`
Indicates that the application process issued a call with an invalid component length for the global title data size. Global title formats are described in "Routing Capabilities" in Chapter 3.

`3024` (CCITT/TTC/NTT/China) `TC_ERR_TOT_LEN_GT_MSU_SIZE`
Indicates that the application process issued a call with an invalid MSU size specified for the global title. Global title formats are described in "Routing Capabilities" in Chapter 3.

`3025` (CCITT/TTC/NTT/China) `TC_ERR_GEN_REJ_COMP`
Indicates a general error in the specification of TCAP components. Review specifications and correct any that are not valid.

`3026` (CCITT/TTC/NTT/China) `TC_ERR_LEN_IND_VALUE_ZERO`
Indicates that the application process issued a call to `ca_put_tc()` with the indicator value length set to zero, which indicates that there is no information defined. Correct the problem by reissuing the call, specifying valid information for the data field.

`3027` (CCITT/CTTC/NTT/China) `TC_ERR_UNI_MSG_NO_COMP`
The application process attempted to send information to another application by calling `ca_put_tc()` with the `t_block_t` structure's `primitive_code` field set to `TC_UNI`. However, the `tc_chp_t` structure's `data` field has a length of `0`, which indicates that there is no information defined. Correct the problem by reissuing the call, specifying valid information for the data field.

`3028` (CCITT/TTC/NTT/China) `TC_ERR_NO_MSU_BUILT`
Indicates a TCAP application process attempted to process MSUs, but there are no MSUs built.

`3029` (CCITT/TTC/NTT/China) `TC_ERR_INV_TSL_STATE`
Indicates a serious condition. Report this error, along with all appropriate debug information, to the Stratus Customer Assistance Center (CAC). This error, which can be returned by the `ca_get_tc()` function, causes an immediate return; the function call is not executed.

`3030` (CCITT/TTC/NTT/China) `TC_ERR_INV_TSL_EVENT`
Indicates a serious condition. Report this error, along with all appropriate debug information, to the Stratus Customer Assistance Center (CAC). This error, which can be returned by the `ca_get_tc()` function, causes an immediate return; the function call is not executed.

`3031` (CCITT/TTC/NTT/China) `TC_ERR_INV_ISM_STATE`
Indicates a serious condition. Report this error, along with all appropriate debug information, to the Stratus CAC. This error, which can be returned by the `ca_get_tc()` function, causes an immediate return; the function call is not executed.

`3032` (CCITT/TTC/NTT/China) `TC_ERR_INV_ISM_EVENT`
Indicates a serious condition. Report this error, along with all appropriate debug information, to the Stratus Customer Assistance Center (CAC). This error, which can be returned by the `ca_get_tc()` or `ca_put_tc()` function, causes an immediate return; the function call is not executed.

`3033` (CCITT/TTC/NTT/China) `TC_ERR_DUPLICATE_BEGIN`
Indicates that the application process is attempting to send a `TC_BEGIN` primitive; however, the `t_block_t` structure's `trans_id` field is not set to the value `-1`, which is necessary for sending that primitive type. Correct the problem by reissuing the call, specifying the value `-1` for the `t_block_t` structure's `trans_id` field. This error can be returned by the `ca_put_tc()` function.

Error messages with numbers 3034 through 3042 are for all network variants.

`3034` `TC_ERR_OUT_OF_TC_USER_ID`
Indicates that a user-supplied function has exhausted its supply of user IDs. This error can be returned by the `ca_get_tc()` function.

`3035 TC_ERR_ISM_NOT_ALLOCATED`
Indicates a serious condition. Report this error, along with all appropriate debug information, to the Stratus CAC. This error, which can be returned by the `ca_put_tc()` function, causes an immediate return; the function call is not executed.

`3036 TC_ERR_ISM_TEQ_OVERFLOW`
Indicates a serious condition. Report this error, along with all appropriate debug information, to the Stratus CAC. This error, which can be returned by the `ca_get_tc()` function, causes an immediate return; the function call is not executed.

`3037 TC_ERR_TSL_TEQ_OVERFLOW`
Indicates a serious condition. Report this error, along with all appropriate debug information, to the Stratus CAC. This error, which can be returned by the `ca_get_tc()` function, causes an immediate return; the function call is not executed.

`3038 TC_ERR_PTR_TO_USF_NOT_SET`
Indicates that the application process called `ca_get_tc()` without specifying a value for the function's `pfunc` parameter. Correct the problem by reissuing the function call, specifying an appropriate value for the `pfunc` parameter.

`3039 TC_ERR_INV_SEQ_CONTROL_VALUE`
Indicates an invalid sequence control value in the application process.

`3040 TC_ERR_INV_PRIORITY_VALUE`
Indicates that the application process called `ca_put_tc()` with an invalid value specified for the `t_block_t` structure's `priority` field. See the description of `ca_put_tc()` in Chapter 6 for a list of valid values.

`3041 TC_ERR_INV_LEN_OF_CONTENTS`
Indicates the set of components exceeds the limits of the XUDT data parameter.

`3042 TC_ERR_INV_EXTND_DATA_PTR`
Indicates that the `tblock` has an invalid pointer to the extended data buffer.

SCCP Errors

This section presents a numeric listing of SCCP errors and their meanings. SCCP errors are returned only in response to a `ca_put_tc()` function call. When an SCCP error occurs, the SINAP/SS7 system does not execute the `ca_put_tc()` function call and it releases the `T_Block`. However, since the invoke state machine (ISM) has already been started, it must time out. Timeout responses are returned to `ca_get_tc()`.

SCCP error numbers are in the range of 4000 through 4999.

`4000 SC_ERR_NOTRANS_NATURE`
Indicates that the application process requires global title translation.

`4001 SC_ERR_NOTRANS_SPECIFIC`
Indicates that a specific application process requires global title translation.

`4002 SC_ERR_SSN_CONGESTION`
Indicates that the application process called `ca_put_tc()` but was unable to complete the call due to subsystem number (SSN) congestion.

`4003 SC_ERR_SSN_FAILURE`
Indicates a subsystem failure occurred.

`4004 SC_ERR_SSN_UNEQUIPPED`
Indicates the application process called `ca_put_tc()`. However, the specified remote SSN was not found or is unequipped.

`4005 SC_ERR_PC_FAILURE`
Indicates a network failure.

`4006 SC_ERR_PC_CONGESTION`
Indicates network congestion.

`4007 SC_ERR_UNQUALIFIED`
Indicates an unqualified error in the system.

`4008 SC_ERR_INV_CTRL`
Indicates that the application process called `ca_put_tc()` with an invalid SCCP control value. Correct the problem by having the application process reissue the call, specifying a valid SCCP control value.

The following messages indicate errors in the SCCP header format.

`4009 SC_INVALID_CLASS`
Indicates that the application process called `ca_put_tc()` with an invalid operating class, which is defined in the `tc_chp_t` structure's `oper_class` field. Correct the problem by reissuing the call, specifying a valid operating class.

`4010 SC_INVALID_RETURN`
Indicates that the application process called `ca_put_tc()` with an invalid SCCP return type, which is defined in the `pa_report_cause` field of the `tc_dhp_t` structure (CCITT/TTC/NTT/China) or the `tc_thp_t` structure (ANSI). Correct the problem by reissuing the call, specifying a valid SCCP return type.

`4011 SC_INVALID_OFFSETS`
Indicates that the application process called `ca_put_tc()` with an invalid offset specified for the calling-party and/or called-party address. Correct the address offset and reissue the call.

`4012 SC_INVALID_NO_CALD_ADDR_PN`
Indicates that the application process called `ca_put_tc()` with an invalid called-party address, which is defined in the `dest_addr` field of the `tc_dhp_t` structure (CCITT/TTC/NTT/China) or the `tc_thp_t` structure (ANSI). Correct the problem by reissuing the call, specifying a valid called-party address.

`4013 SC_INVALID_MESSAGE_TYPE`
Indicates that the application process called `ca_put_tc()` with an invalid message type, which is defined in the `tc_chp_t` structure's `comp_type` field. Correct the problem by reissuing the call, specifying a valid message type.

`4014 SC_INVALID_ADDRESS_LENGTH`
Indicates that the application process called `ca_put_tc()` with a calling-party or called-party address whose length exceeds the maximum allowed. (These addresses are defined in the `orig_addr` and `dest_addr` fields, respectively, of the `tc_dhp_t` structure (CCITT/TTC/NTT/China) or the `tc_thp_t` structure (ANSI).) Correct the problem by reissuing the call, specifying a valid address of acceptable length.

`4015 SC_INVALID_ADDRESS`
Indicates that the application process called `ca_put_tc()` with an invalid calling-party and/or called-party address. (These addresses are defined in the `orig_addr` and `dest_addr` fields, respectively, of the `tc_dhp_t` structure (CCITT/TTC/NTT/China) or the `tc_thp_t` structure (ANSI).) Either the address format is incorrect or the address itself is invalid. Correct the problem by reissuing the call, specifying a valid address in the correct format.

`4016 SC_INVALID_SSN`
Indicates that the application process called `ca_put_tc()` with an invalid subsystem number. Correct the problem by reissuing the call, specifying a valid subsystem number.

The following message tells CASL to discard an outbound message:

`4017 SC_ERR_DISCARD`
Indicates that the application process called `ca_put_tc()` to send an outbound MSU; however, the SCCP was unable to route the MSU and therefore discarded it.

The following messages indicate SCCP connection-oriented control (SCOC) class 2 and class 3 function errors.

`4025 SC_ERR_SID_NO_CID`
Indicates that no connection ID is available for the calling process.

`4026 SC_ERR_SID_TOO_MANY`
Indicates that the calling application or subsystem number is using too many local reference memory (LRM) segments.

`4027 SC_ERR_SID_CID_NG`
Indicates that the specified connection ID is out of range for the calling process.

`4028 SC_ERR_SID_NOT_IDLE`
Indicates that the application process called `ca_rel_sccpid` with the local reference memory (LRM) not in the idle state.

`4029 SC_ERR_SID_NO_LRN`
Indicates that no source local reference number (LRN) is available for the calling process.

`4030 SC_ERR_CID_REPLY`
Indicates that the SINAP/SS7 system cannot send the connection ID reply to the requesting process.

`4031 SC_ERR_CON_REQ`
Indicates that the SINAP/SS7 system cannot send `N_CONNECT_REQ` to the calling process.

`4032 SC_ERR_CID_HUNT`
Indicates that the SINAP/SS7 system cannot locate the LRM or LRN for the calling process.

`4033 SC_ERR_CON_CON`
Indicates that the SINAP/SS7 system cannot send `N_CONNECT_CON` to the calling process.

`4034 SC_ERR_CON_REF`
Indicates that the SINAP/SS7 system cannot send `N_DISCONNECT_IND` to the calling process.

`4035 SC_ERR_RES_REQ`
Indicates that the SINAP/SS7 system cannot send `N_RESET_REQ` to the calling process.

`4036 SC_ERR_INV_MSU`
Indicates that SCOC received an invalid MSU type and cannot process it..

`4037 SC_ERR_INV_MSU_CTRL`
Indicates that the SCOC received an invalid `sccp_ctrl` structure for the MSU.

`4038 SC_ERR_INV_MSU_NI`
Indicates that the SCOC received an invalid `sccp_ctrl` structure for the MSU.

`4039 SC_ERR_CON_REQ_INV`
Indicates that the application process sent an invalid connection request data format.

`4040 SC_ERR_CON_REQ_NG`
Indicates that the SCOC cannot sent the `CON_REQ` structure to the SINAP/SS7 system.

`4041 SC_ERR_INV_MSU_NI`
Indicates that the SCOC received an invalid `CON_IND` structure.

`4042 SC_ERR_INV_PRIM`
Indicates that the SCOC received an unimplemented primitive.

`4043 SC_ERR_CON_CON_INV`
Indicates that the SCOC received an invalid `CON_CON` function call.

`4044 SC_ERR_DIS_IND_INV`
Indicates that the SCOC received an invalid `DIS_CON` structure.

`4045 SC_ERR_RES_IND_INV`
Indicates that the SCOC received an invalid `RES_IND` structure.

`4046 SC_ERR_RES_CON_INV`
Indicates that the SCOC received an invalid `RES_CON` structure.

`4047 SC_ERR_CON_CON_NG`
Indicates that the SCOC cannot send the `RES_IND` structure to the SINAP/SS7 system.

`4048 SC_ERR_CID_NO_SSN`
Indicates that no SSN was specified in the `get_connid` field.

`4049 SC_ERR_CON_CON_BAD`
Indicates that the `CON_CON` structure contains invalid data.

`4050 SC_ERR_DIS_IND_NG`
Indicates that the SCOC cannot send the `N_DISCONNECT_IND` field to the SINAP/SS7 system.

`4051 SC_ERR_DIS_IND_NG1`
Indicates that the SCOC cannot send the `N_DISCONNECT_IND` field to the user.

`4052 SC_ERR_RES_REQ_NG`
Indicates that the SCOC cannot send the `N_RESET_REQ` structure to the SINAP/SS7 system.

`4053 SC_ERR_RES_REQ_BAD`
Indicates that the `RES_REQ` structure contains invalid data.

`4054 SC_ERR_RES_CON_NG`
Indicates that the SCOC cannot send the `RES_CON` structure to the SINAP/SS7 system.

`4055 SC_ERR_RES_CON_BAD`
Indicates that the `RES_CON` structure contains invalid data.

`4056 SC_ERR_RES_CON_NG1`
Indicates that the SCOC cannot send the `RES_CON` structure to the user.

`4057 SC_ERR_REL_NG`
Indicates that the SCOC cannot send the `N_DISCONNECT` field to the SINAP/SS7 system.

`4058 SC_ERR_REL_BAD`
Indicates that the `N_DISCONNECT_REQ` field contains invalid data.

`4059 SC_ERR_CID_SSN_NG`
Indicates that no SSN will be available when the `conn_id` is released..

`4060 SC_ERR_IT_BAD`
Indicates that the `ca_put_tc` function call received an IT with an invalid local reference number (LRN).

`4061 SC_ERR_IT_SEND_NG`
Indicates that the `ca_put_tc` function call cannot send an IT MSU.

`4062 SC_ERR_RSEQNO_NG`
Indicates that the `ca_put_tc` function call received a sequence number that was not valid.

`4063 SC_ERR_RLSD_SEND_NG`
Indicates that the `ca_put_tc` function call cannot send the RLSD MSU.

`4064 SC_ERR_REF_NG`
Indicates that the `ca_put_tc` function call cannot send the `CON_REF` structure to the SINAP/SS7 system.

MTP Errors

This section numerically lists and describes MTP error messages. These messages are returned only in response to a `ca_put_tc()` function call. MTP error messages number in the range of 5000 to 5999.

`5002 L3_ERR_HOLDQUE`
The application process called `ca_put_tc()` but the MSU (`M_Block`) is being held due to rerouting.

`5003 L3_ERR_NO_ROUTING_FOR_DPC`
The application process called `ca_put_tc()` but the SINAP/SS7 system was unable to establish a route to the specified DPC. Check that your SINAP/SS7 configuration provides access to this DPC, make any necessary corrections to your configuration, and reissue the

call.

`5004 L3_ERR_INACCESSIBLE_DPC`
The application process called `ca_put_tc()` but the route(s) to the specified DPC are not currently accessible. You can reissue the call.

`5007 L3_ERR_NO_HOLD_MEMORY`
The application process called `ca_put_tc()` but no memory was available to hold the MSU during rerouting. Redefine memory allocations and reissue the call.

`5012 L3_DR_DISCARDED_OFFLINE`
The application process called `ca_put_tc()` but could not process the MSUs because the SINAP/SS7 system was offline.

`5013 L3_DR_DISCARDED_TXQFULL`
The application process called `ca_put_tc()` but could not process the MSU because the TXQ was full.

`5014 L3_ERR_DT_NOT_PROVISIONED`
The application process called `ca_put_tc()` but could not process the MSU because the discrimination table was not provisioned.

`5015 L3_ERR_DISCARD`
The application process called `ca_put_tc()` but the link is congested and the MSU can not be sent; therefore, it is discarded.

Built-In Test Environment (BITE) Errors

BITE errors number in the range of 6000 through 6999. There are currently no CASL errors associated with the BITE system.

Application Errors

Application errors number in the range of 7000 through 7999. There are currently no CASL errors associated with the applications themselves.

Index

D

E

F

K

N

O

Q

R

S

T

U

V

X

Z