Dialogic
Making Innovation Thrive™

Dialogic® NaturalAccess™ TCAP Layer Developer's Reference Manual

July 2009 64-0462-01

www.dialogic.com

Copyright and legal notices

Revision history

Revision	Release date	Notes
9000-6469-10	July 1997	TCAP preliminary release A.0.2
9000-6469-11	October 1997	TCAP preliminary release A.0.3
9000-6469-12	September 1998	GJG
9000-6469-13	March 1999	GJG
9000-6469-14	March 2000	GJG
9000-6469-15	November 2000	GJG, SS7 3.6
9000-6469-16	August 2001	GJG, SS7 3.8 beta
9000-6469-17	November 2003	SRG, SS7 4.0
9000-6469-18	April 2005	SRG, SS7 4.2
9000-6469-19	July 2006	LBZ, SS7 4.3
9000-6469-20	September 2008	LBG, SS7 5.0
64-0462-01	July 2009	LBG, SS7 5.1
Last modified: July 7, 2009		

Refer to www.dialogic.com for product updates and for information about support policies, warranty information, and service offerings.

Table Of Contents

Chapter 1: Introduction 9

Chapter 2: SS7 overview 11
SS7 architecture 11
TX board components 12
Host components 12
TCAP task 13

Chapter 3: TCAP programming model 15
Programming model overview 15
TCAP service users 15
Entity and instance IDs 16
NMS TCAP functions 16
Service functions 16
Management functions 17
Queues and contexts 18
Single-context, single-queue model 18
Multiple-context, single-queue model 19
Multiple-context, multiple-queue model 20
SCCP quality of service (QOS) 21
TCAP transactions 22
ANSI transaction types 22
ITU-T transaction types 22
TCAP components 23
Message lengths and segmentation 24
Multiple-threaded considerations 24
Transaction checkpointing 25
Congestion control 26
Outbound congestion 26
Inbound congestion 29
TCAP configuration 30
SCCP addressing and routing 31
Routing by point code and subsystem number 31
Routing by global title 31
SCCP address override 32
Global title translation 33
Setting function parameters 33
Setting the SCCP configuration 34
Setting variations in global title translation 35
Status and notify indications 36
Dialog IDs 36

Chapter 4: Using the TCAP service 37
Setting up the Natural Access environment 37
Initializing the Natural Access environment 37
Creating queues and contexts 37
Binding to the TCAP service 38
Receiving TCAP service events 39
Handling redundancy events 40
Generating TCAP transactions 40

Simple request and response transaction 41
Conversational linked transaction....42
Handling abnormal conditions 44
Invalid transaction portions....44
Transaction inactivity timeouts 45
Invalid component in a begin or query message....46
Invalid component in a continue or conversation message....47
Invalid component in an end or response message 48
Invoke time-outs (ITU-T only)....49
Invalid component in a multiple component message....49
Signaling point and subsystem status....50
Coordinated state change 50
Subsystem state changes 51
Remote signaling point failures....51
Tracing function calls and events 52

Chapter 5: TCAP service function reference 53
TCAP service function summary....53
Using the TCAP service function reference....54
TCAPAddComp 55
TCAPCoordReq....56
TCAPCoordResp 57
TCAPGetApiStats....58
TCAPGetComp 59
TCAPInitTrans....61
TCAPRetainTrans....62
TCAPRetrieveMessage 63
TCAPStateReq....64
TCAPTransRqst 65

Chapter 6: TCAP management function reference....67
TCAP management function summary 67
Using the TCAP management function reference 68
TCAPAlarmControl 69
TCAPGenCfg 70
TCAPGenStatus....71
TCAPGetGenCfg 72
TCAPGetSapCfg 73
TCAPInitGenCfg 74
TCAPInitMgmtAPI 76
TCAPInitSapCfg....77
TCAPSapCfg 80
TCAPSapStats....81
TCAPTermMgmtAPI....83
TCAPTraceControl....84

Chapter 7: Demonstration programs and utilities 85
Summary of the demonstration programs and utilities 85
Request and response transaction: find800 86
800 number server 88
800 number client 88
Troubleshooting 89
Using the TCAP ITU-T protocol....89

Adding subsystem numbers 89
TCAP configuration utility: tcapcfg 90
TCAP layer status: tcapmgr 91

Chapter 8: Parameter and event structure overview 93
Data types 93
Point codes....93
TCAP octet strings 93
TCAP component IDs 94
Global titles....94

Chapter 9: Common data structures 97
Common data structures summary 97
SCCP address structure....98
SCCP quality of service (QOS) structure 100
TCAP transaction information structure....101
TCAP dialog section structure....103

Chapter 10: Component data structures....105
Component structure format....105
ANSI component structure....106
TcapAnsiOpcode....107
TcapAnsiErrcode....109
TcapAnsiPrbcode 109
ITU-T component structure....111

Chapter 11: Incoming message event structures 115
Messages overview....115
General receive information block structure 115
TCAP coordinated event structure....116
Signaling point status event structure 117
Subsystem status event structure....118
TCAP notification event structure....119
TCAP status event structure....120
TCAP transaction data event structure....121

1 Introduction

The *Dialogic® NaturalAccess™ TCAP Layer Developer's Reference Manual* explains how to implement the SS7 TCAP layer using NaturalAccess™ TCAP. This manual explains how to create applications using NaturalAccess™ TCAP and presents a detailed specification of its messages and functions.

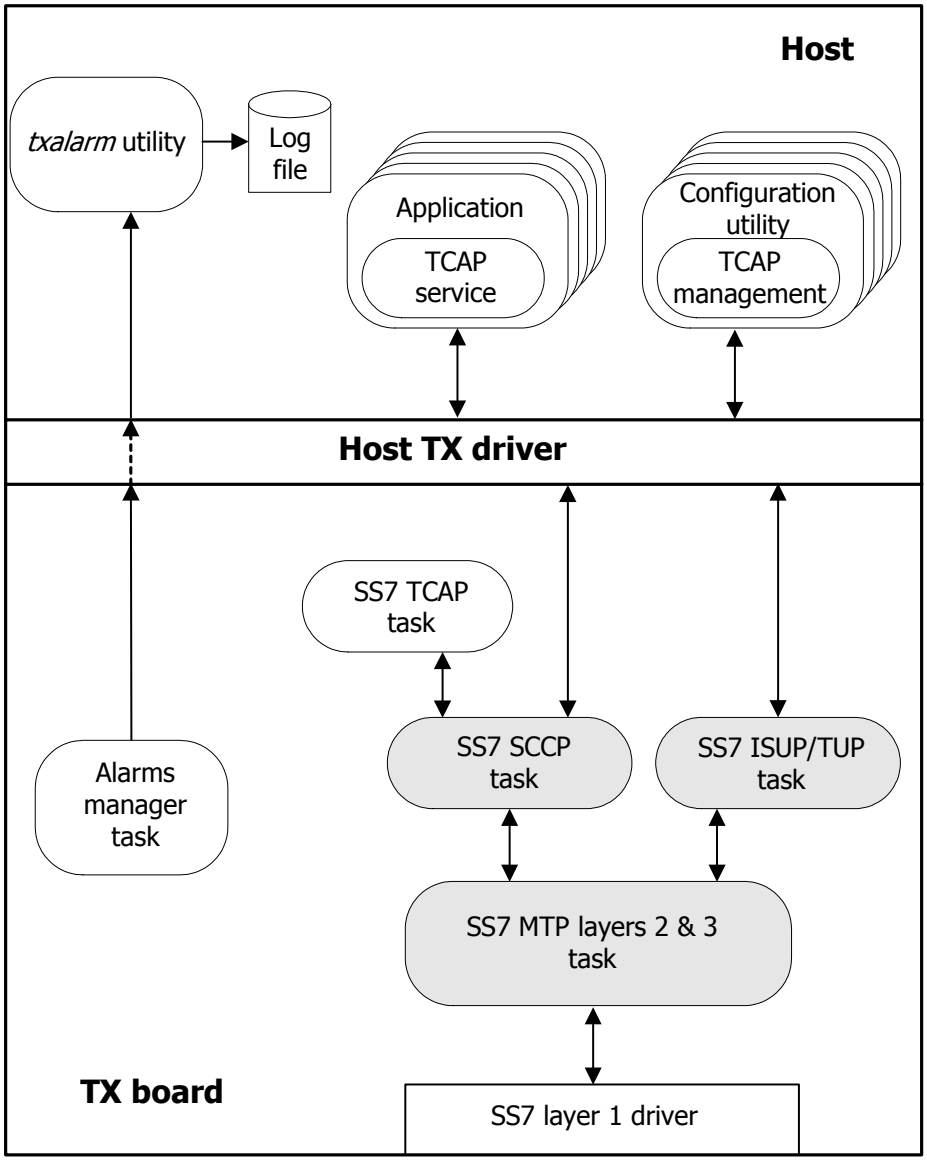
Note: The product(s) to which this document pertains is/are among those sold by NMS Communications Corporation ("NMS") to Dialogic Corporation ("Dialogic") in December 2008. Certain terminology relating to the product(s) has been changed, whereas other terminology has been retained for consistency and ease of reference. For the changed terminology relating to the product(s), below is a table indicating the "New Terminology" and the "Former Terminology". The respective terminologies can be equated to each other to the extent that either/both appear within this document.

Former terminology	Current terminology
NMS SS7	Dialogic® NaturalAccess™ Signaling Software
Natural Access	Dialogic® NaturalAccess™ Software
NMS TCAP	Dialogic® NaturalAccess™ TCAP Layer

2 SS7 overview

SS7 architecture

The following illustration shows the SS7 software architecture in a typical system with separate host applications handling transactions, the system configuration, and system alarms:



TX board components

The TX board consists of the following components:

- TCAP task that implements the SS7 TCAP layer.
- SCCP task that implements the SS7 SCCP layer.
- MTP task that implements the SS7 MTP 2 (data link) layer and the MTP 3 (network) layer.
- Optional ISUP/TUP task that implements the SS7 ISUP/TUP layer.
- TX alarms manager task that collects unsolicited alarms (status changes) generated by the SS7 tasks and forwards them to the host for application-specific alarm processing.

Host components

The host consists of the following components:

- A TX driver for the native host operating system that provides low-level access to the TX board from the host.
- Functions that provide the application with a high-level interface to the TCAP layer services.
- Functions that provide the application with a high-level interface to the TCAP management layer services.
- An alarm collector process for capturing alarms and saving them to a text file. The alarm collector (*txalarm*) is provided in both executable and source form. The source can be used as an example for developers who want to integrate the TX alarms into their own alarm monitoring system.
- Configuration utilities (one for each SS7 layer) that read the SS7 configuration file(s) and load the configurations to the TX processor tasks at system startup. The TCAP configuration utility (*tcapcfg*) is provided in both executable and source form. The source code can be used as an example for developers who want to integrate the TCAP configuration into their own configuration management system.
- The TCAP manager utility (*tcapmgr*) provides a command line interface from which alarm levels can be set, buffers can be traced, and TCAP statistics can be viewed and reset.

TCAP task

The TCAP task provides the following services on behalf of applications:

- Multiple outstanding transactions per application/subsystem number
- Assembling of application components (operation invokes and replies) and parameters into TCAP messages
- Association of replies with invokes
- Optional timing for replies to invokes (ITU-T only)
- Handling of abnormal conditions: protocol errors, timeouts, aborted transactions
- Use of ITU-T standard TCAP with ANSI-standard MTP/SCCP stack and vice versa

The TCAP task uses the SCCP connectionless transport service. The TCAP task also makes the following SCCP layer services available to the application:

- Addressing by point code/subsystem number and/or global title
- Subsystem and point code status change indications
- Replicated subsystems with coordinated state change

3 TCAP programming model

Programming model overview

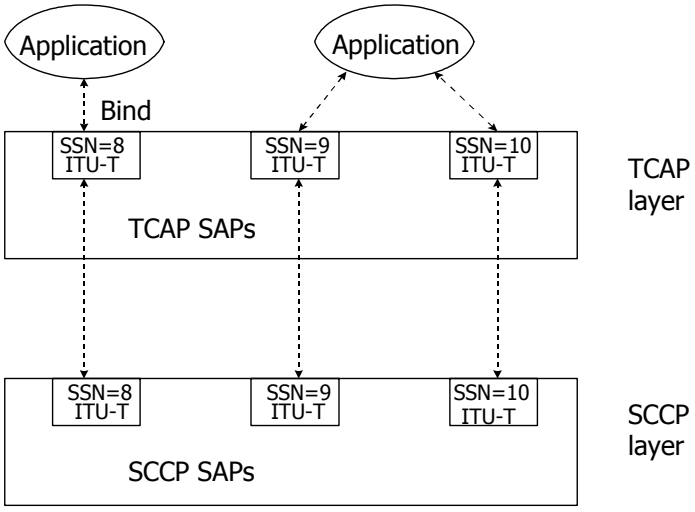
NMS TCAP consists of a set of function calls that provide access to the TCAP layer operations, and a set of events that notify the application of incoming messages, network status, and message delivery errors. NMS TCAP also performs the byte ordering translation, where necessary, between application processor (little endian) byte order and network (or big endian) byte order.

NMS TCAP is implemented as a Natural Access service. Natural Access is a development environment for telephony and signaling applications that provides a standard application programming interface for services, such as signaling protocol stacks, independent of the underlying hardware. Understanding the basic Natural Access programming concepts such as services, queues, contexts, and asynchronous events is critical to developing applications that use the TCAP service. Refer to the *Natural Access Developer's Reference Manual* for more information.

TCAP service users

NMS TCAP supports one or more applications with service access points, or SAPs. One SAP is defined for each application that uses the TCAP service. At initialization, applications bind to a particular SAP by specifying the SAP ID. Each user SAP is associated with a single SCCP subsystem number. All TCAP messages destined for a particular subsystem number are routed to the application bound to the SAP associated with that subsystem number.

SAPs are shown in the following illustration:



Note: The number of SAPs and the characteristics of each SAP are specified at TCAP configuration time. Refer to the *NMS SS7 Configuration Manual* for more information.

Entity and instance IDs

Each application must have a unique entity and instance ID for routing messages among the processes in the system. Entity IDs are single byte values in the range of 0x00 - 0xFF, assigned by the application developer. Entity IDs are allocated as follows:

Range	Usage
0x00 - 0x1F 0x80 - 0xFF	Reserved for use by system utilities, configuration utilities, and management utilities.
0x20 - 0x7F	Available for use by applications.

Instance IDs identify the processor on which the entity executes. The host is always processor 0 (zero). Therefore, all host-resident TCAP applications must be coded to 0 (zero). All tasks on TX board number 1 receive an instance ID of 1. All tasks on TX board number 2 receive an instance ID of 2, and so on.

NMS TCAP functions

NMS TCAP provides two sets of functions:

- Service functions
- Management functions

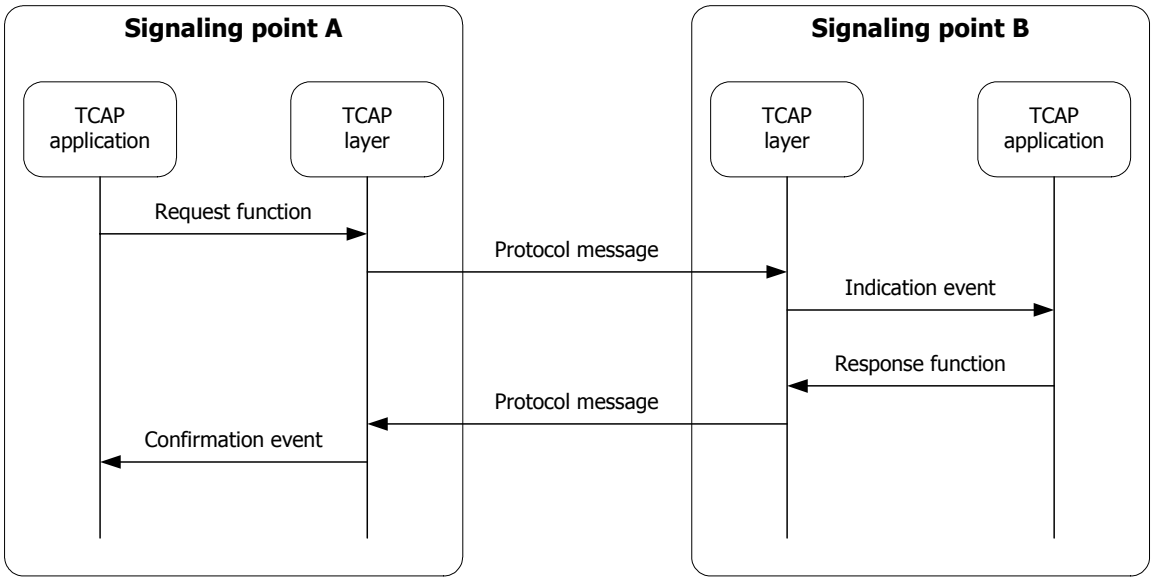
Service functions

The TCAP service functions provide the application access to the TCAP layer services. Applications invoke TCAP services by calling TCAP request functions that generally result in a TCAP message to a remote exchange, or signaling point (SP). Request function parameters are converted to messages and sent through the device driver to the TCAP task.

The TCAP requests from the remote signaling points are presented to the application at the receiving side as indications.

The receiving application then issues a reply to the originating signaling point by invoking the appropriate TCAP response function. The response function is typically translated by the SCCP layer into a protocol message back to the originating signaling point. That response is presented back to the application as a confirmation.

This communication model is shown in the following illustration. Some operations, such as sending unit data, include only the request or indication steps. These operations are called unconfirmed operations.



All TCAP service functions are asynchronous. Completion of the function implies only that the function was successfully initiated (a request message was queued to the SCCP task). Errors detected by the SCCP task result in asynchronous status indications being sent to the application. Successfully delivered requests generally result in no notification to the application until the far end takes some corresponding action such as responding to a transaction invoke component with a return-result component.

Indication and confirmation messages, as well as status messages from the local TCAP layer, are passed to application processes as asynchronous events. All events for a particular user SAP (subsystem) are delivered through the associated Natural Access queue. For information about queues, refer to the *Natural Access Developer's Reference Manual*.

Applications detect that an event is pending through a call to **ctaWaitEvent**. The application retrieves the event data through a function that translates the event parameters from SS7 TCAP raw format to API format.

For more information, refer to the *TCAP service function summary* on page 53.

Management functions

Unlike the TCAP service functions that send and receive messages asynchronously, each TCAP management function generates a request followed immediately by a response from the TX board. TCAP management functions block the calling application waiting for this response (typically a few hundred milliseconds) and return an indication as to whether or not an action completed successfully. For this reason, the TCAP management functions are typically used by one or more management applications, separate from the applications that use the TCAP service functions. TCAP management is packaged as a separate library with its own interface header files.

For more information, refer to the *TCAP management function summary* on page 67.

Queues and contexts

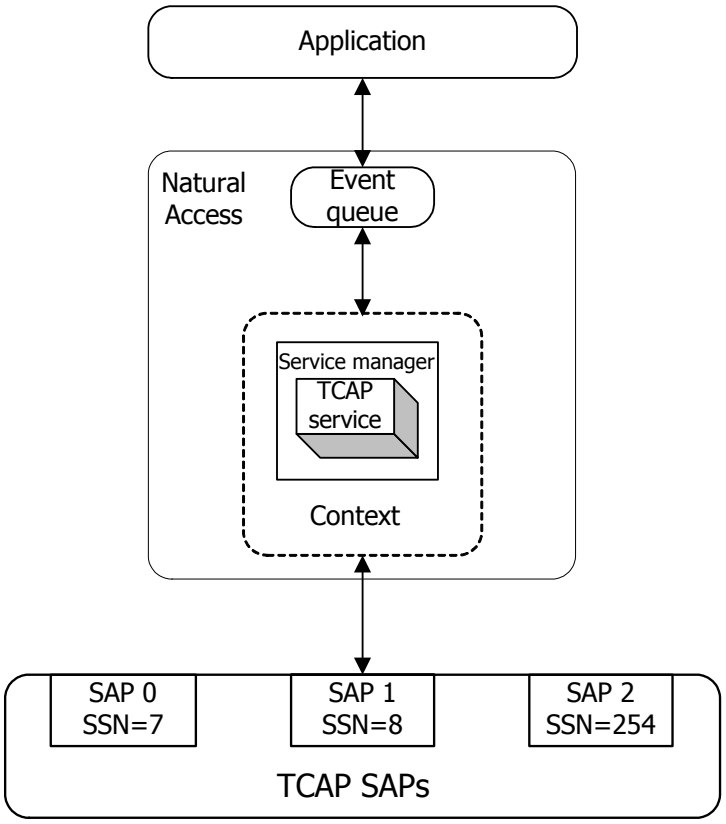
Natural Access organizes services and their associated resources around a processing object known as a context. Each instance of an application binding to a TCAP service access point is a unique Natural Access context. Contexts are created with **ctaCreateContext**.

All events and messages from the TCAP service are delivered to the application through a Natural Access queue object. Queues are created with **ctaCreateQueue**. Each context is associated with a single queue through which all events and messages belonging to that context are distributed. More than one context can be assigned to the same queue.

Different application programming models are possible depending on how many TCAP service access points (how many TCAP subsystems) are implemented by the application and how the application is organized.

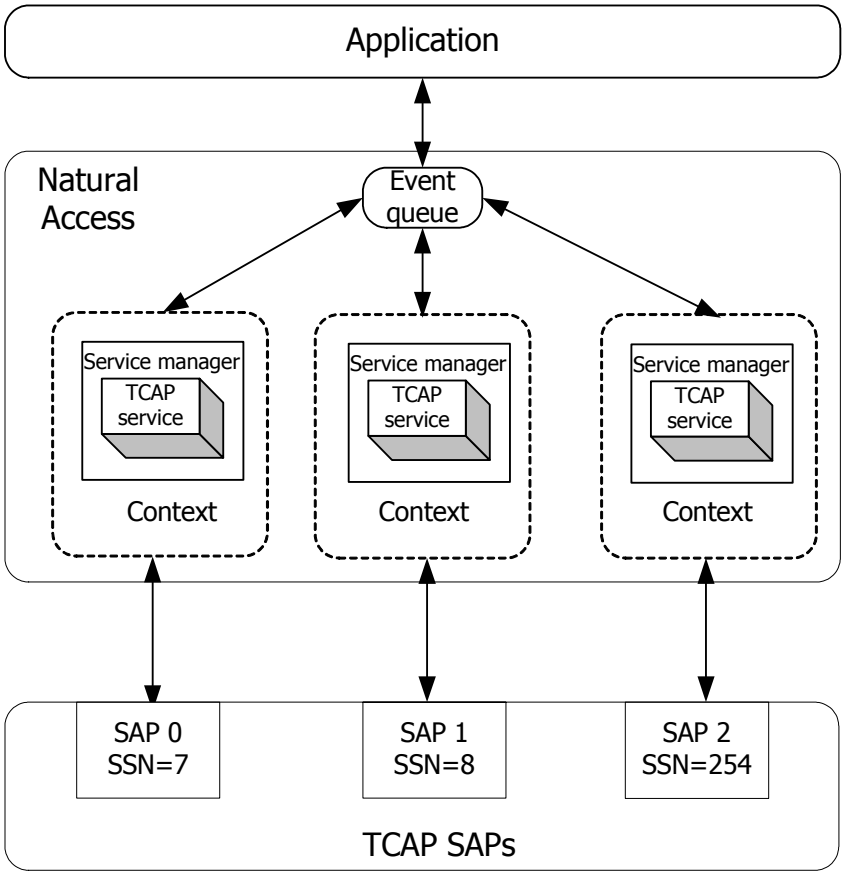
Single-context, single-queue model

An application that uses a single TCAP service access point uses a single-context, single-queue model as shown in the following illustration:



Multiple-context, single-queue model

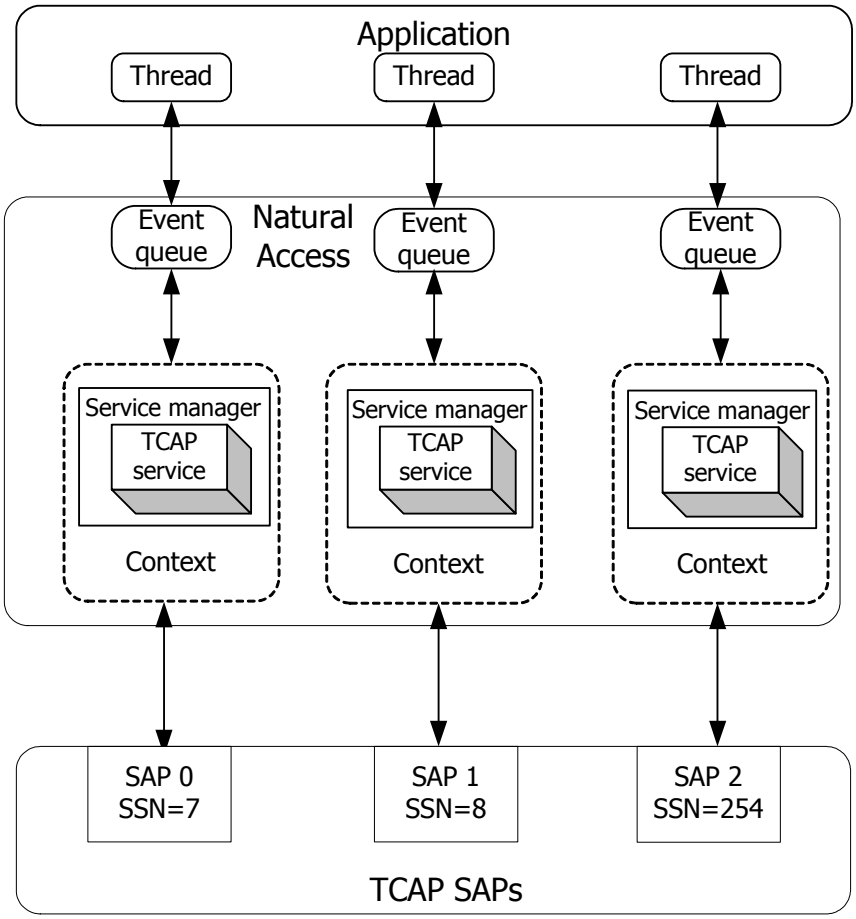
For a single-threaded application that uses multiple TCAP service access points (implements multiple subsystems), a multiple-context, single-queue model is recommended (as shown in the following illustration). In this case, the application has a single event loop with events from all service access points delivered through the same queue. The application determines which service access point a particular event is associated with from a service user ID (suID) value returned with each event.



Multiple-context, multiple-queue model

For multiple-threaded applications using multiple TCAP service access points (one per thread), a multiple-context, multiple-queue model is recommended (as shown in the following illustration). In this case, each thread has its own event loop and receives only the events associated with a service access point on its Natural Access queue.

Note: For this programming model, each thread or event queue must be assigned its own entity ID. The entity ID must be unique among all applications on that host accessing any of the SS7 services.



SCCP quality of service (QOS)

The TCAP layer uses the SCCP connectionless transport service to deliver transaction messages to their destinations. With this service, three additional quality of service options are available to the application:

Quality of service option	Description
Message priority	The application can assign a priority of 0 (lowest priority) through 3 (highest priority).

Refer to the *NMS SCCP Developer's Reference Manual* for information about SCCP.

TCAP transactions

This topic provides the following information about TCAP transactions:

- ANSI and ITU-T transaction types
- TCAP components
- Message lengths and segmentation
- Multiple-threaded considerations
- Transaction checkpointing

ANSI transaction types

The TCAP layer supports the following ANSI transaction types (also called package types):

Transaction type	Description
Unidirectional	Sends information in one direction only, with no reply expected. No TCAP transaction is established.
Query with permission	Initiates a TCAP transaction and gives the destination node permission to end the transaction.
Query without permission	Initiates a TCAP transaction and notifies the destination node that it cannot end the transaction.
Conversation with permission	Continues a TCAP transaction and gives the destination node permission to end the transaction.
Conversation without permission	Continues a TCAP transaction and notifies the destination node that it cannot end the transaction.
Response	Ends a TCAP transaction.
Abort	Terminates a TCAP transaction without sending pending components.

ITU-T transaction types

The TCAP layer supports the following ITU-T transaction types:

Transaction type	Description
Unidirectional	Sends information in one direction only, with no reply expected. No TCAP transaction is established.
	Initiates a TCAP transaction.
Continue transaction	Continues a TCAP transaction and transfers data in either direction.
End transaction	Ends a TCAP transaction.
Abort	Terminates a TCAP transaction without sending pending components.

TCAP components

TCAP transactions are composed of components that are remote operation invocations, or invokes, and responses to invokes. Each TCAP message (begin, continue, end, query with permission, response, and so on) is composed of zero or more components. Each component can optionally have application-specific parameters associated with it. Component parameters are not interpreted by the TCAP layer but are passed on to the destination application.

The ANSI and ITU-T standards define a fairly compatible set of component types:

Component type	Description
Invoke [Last | Not Last]	Invokes a remote operation. This component type applies to ANSI and ITU-T standards.
Return Result [Last | Not Last]	Reply for a successful operation invocation. This component type applies to ANSI and ITU-T standards.
Return Error	Reply for an unsuccessful operation invocation.
Reject	Reports the receipt and rejection of an incorrect transaction package or component. This can be generated by either the TCAP layer itself or by the application.
Cancel	Local operation only. Cancels an outstanding invoke between the application and the TCAP layer, but does not notify the far end.

Each invoke component is identified by a unique invoke ID that was assigned by the application that originated the invoke. The invoke ID is returned in the response, and allows the originator to correlate the reply with the invoke operation to which it belongs.

A receiver of an invoke component can also generate a linked invoke in response to the original invoke received. An example of this is when collecting more information from the sending node before generating the response to the original invoke. The linked invoke carries its own unique invoke ID plus the invoke ID of the original invoke, called the correlation ID in ANSI or the linked ID in ITU-T. It allows the originating application to associate the linked invoke with its original invoke.

In ITU-T, an invoke operation is assigned by the application to one of four classes, numbered 1 through 4. The class designation specifies under what conditions a response is expected, and determines what a timeout implies about the success or failure of the operation.

Class type	Description
1	Both successes and failures are reported. A timeout is an abnormal failure.
2	Only failure is reported. A timeout implies successful completion.
3	Only success is reported. A timeout implies a failed operation.
4	Neither success or failure is reported. There is no interpretation of timeout.

Message lengths and segmentation

The maximum size of a TCAP transaction, including transaction part, component parts, and application parameters is approximately 254 bytes, possibly less if global titles are used in SCCP addresses.

If an application has transactions (invokes or responses) that exceed the maximum size, it must provide its own segmentation and reassembly, using either the last/not last designation for invokes (ANSI only) or responses, or some other application-specific solution.

Multiple-threaded considerations

In a multiple-threaded application, any thread can generate a transaction request when initiating a new transaction or when responding to a transaction initiated by a far SP. However, there should be a single receiver thread that receives all asynchronous events from the TCAP layer (new transaction invocations, transaction responses, and status indications) and routes them to the proper transaction thread based on the user dialog ID or other transaction information.

Consider, for example, an application that consisted of a main thread, which spawned a child transaction thread whenever it wanted to initiate a new transaction. The transaction thread would be responsible for initiating the transaction request, collecting the response, and taking the appropriate action. The application could have many simultaneous transactions, and transaction threads, active at any given time.

Multiple transaction threads could generate the transaction requests in any order. The requests would be processed by the TCAP layer in the order that they were received. If each transaction thread then called **TCAPRetrieveMessage** to retrieve the response to its transaction request, there would be no guarantee that the TCAP message returned by **TCAPRetrieveMessage** would be associated with the calling thread's transaction. The first thread to call **TCAPRetrieveMessage** when a message is pending would receive the first pending message, regardless of which transaction it was associated with. The incoming message could be a new transaction request from a far SP or a network status indication message.

Instead, the main thread itself or a separate child receiver thread should be the only caller of **TCAPRetrieveMessage**. It can then analyze each incoming event and route it to the proper child transaction thread or, in the case of a new incoming transaction, create a new child transaction thread to handle the incoming transaction request.

Note: In a multiple-threaded application similar to the one described here, call **ctaInitialize** from the parent thread before any of the other functions are called by any of the other threads.

Transaction checkpointing

If two TX boards are configured as a redundant pair, TCAP transaction checkpointing can be used. In this configuration, one board is designated as the primary and the other is designated as the backup. The TCAP task on the backup board is ready to take over the TCAP service if the primary board fails or is taken out of service.

To enable the backup TCAP task to take over the TCAP service, the primary and backup TX boards are connected by a private Ethernet connection. The primary TCAP task sends transaction information to the backup TCAP task, which is known as transaction checkpointing. If the primary board fails, or is taken out of service, the backup TCAP task has a complete list of open TCAP transactions so that it can properly handle TCAP traffic.

Transaction checkpointing is performed automatically, but can be controlled by the TCAP application. A default checkpointing behavior can be set in the TCAP configuration file for each service access point. The DEFAULT_CHECKPOINT field in the SAP section of the TCAP configuration file can be set to one of the following values:

Value	Description
CHKPT_ALL	Checkpoint all transactions.
CHKPT_SEND	Checkpoint only transactions initiated by the application.
CHKPT_NONE	Do not checkpoint any transactions.

A TCAP application can override the default checkpointing behavior for a single transaction by setting the chkpt field in the TCAPTransInfo structure to one of the following values:

Value	Description
TCAP_NO_CHKPT	Transaction is not checkpointed.
TCAP_CHKPT	Checkpoint this transaction.
TCAP_CHKPT_DEFAULT	Use the default checkpoint value.

A transaction can be checkpointed at any time during the transaction lifetime. For example, if a begin message is received and the transaction is not checkpointed (the default is set to CHKPT_SEND or CHKPT_NONE), the transaction can be checkpointed when the application sends a reply (continue) message by setting the chkpt field to TCAP_CHKPT. All further transaction information is checkpointed.

Congestion control

Understanding the TCAP service congestion control mechanisms and developing an effective application congestion control strategy is a critical step in developing a reliable system.

The TCAP service implements a four-level congestion control strategy. Level zero (0) indicates that TCAP is not congested and level three reflects the most congested state. The TCAP service further distinguishes between outbound and inbound congestion and maintains a separate congestion level for each direction.

Congestion type	Description
Outbound	From the application and towards the network.
Inbound	From the network and towards the application.

Congestion control mechanisms include notifications to applications of congestion conditions and congestion control actions within the TCAP layer itself. Congestion notifications allow the application to take some corrective action, such as reducing the traffic load that it generates, before the congestion becomes severe and impacts the operation of the service. Congestion control actions within TCAP maintain the operation of the SS7 stack (possibly at a reduced capacity), including the operation of other SS7 layers and/or applications, during periods of congestion.

Outbound congestion

Outbound congestion in TCAP occurs when the:

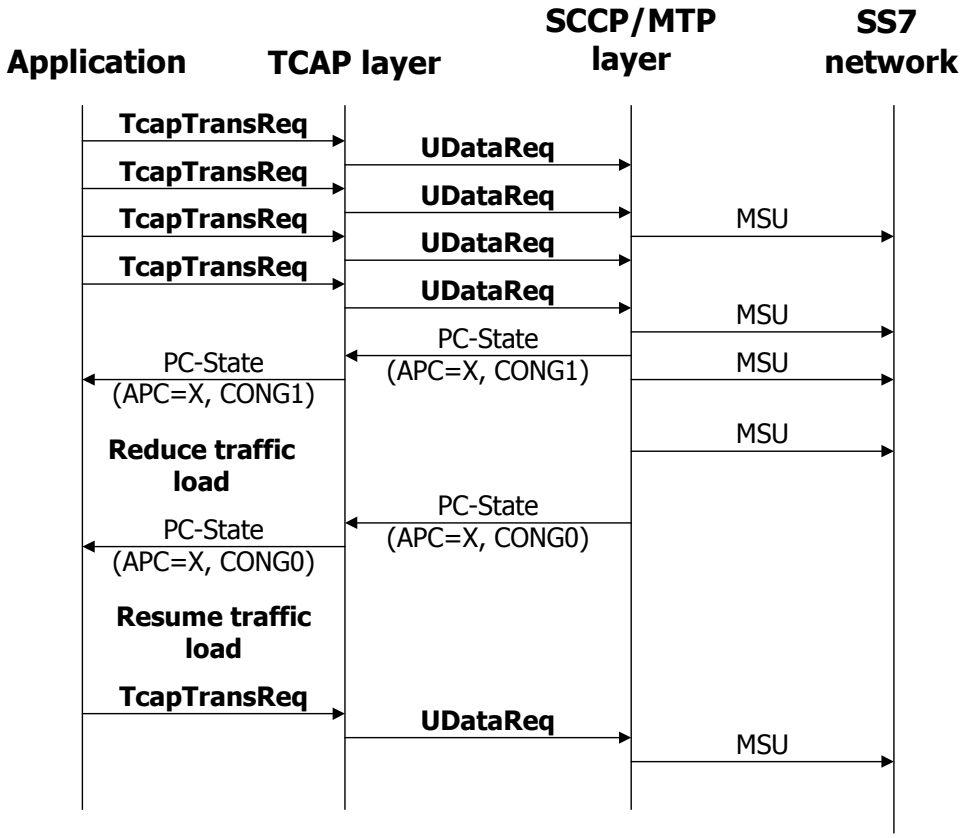
- Application generates TCAP traffic at a rate greater than the capacity of the SS7 links or downstream network, resulting in network overload.
- Application generates TCAP requests faster than they can be processed by the TCAP layer, resulting in the TCAP service send queue building up beyond pre-determined thresholds (TCAP service congestion).
- Overall memory available to the TCAP layer drops below pre-determined thresholds (TCAP layer congestion).

Network overload

Network overload occurs when the MTP layer's outbound queues build up beyond configured limits due to:

- A traffic load that exceeds the capacity of the available signaling links
- Receipt of a transfer controlled message (TFC) regarding a congested destination

In either case, the application receives a TCAP PC-STATE indication containing the affected pointed code and the current congestion level. The application should reduce its traffic load toward the affected destination until the congestion abates. How the reduction is accomplished is up to the application. The TCAP layer itself takes no action to prevent further congestion. The following illustration shows a network overload condition.



In ANSI networks and in other national networks employing multiple congestion levels, the application must not generate any new traffic towards the affected destination with a priority lower than the current destination congestion level, as it will be discarded at the MTP layer.

For the international signaling network, and other ITU-based networks without multiple congestion priorities, it is important for the application to reduce the traffic load toward the affected destination, as the MTP layer only discards outgoing packets in cases of excessive queuing of traffic to congested signaling links. If the application fails to reduce its traffic load toward the congested destination, it can escalate the congestion condition.

When the network overload condition ceases, the application receives a TCAP PC-STATE indication containing the affected point code and a status value of SP_CONG_OFF, indicating that the application can resume normal traffic towards the affected destination.

TCAP service congestion

TCAP service congestion occurs when the application generates traffic faster than it can be accepted by the TCAP layer, resulting in the TCAP service transmission queue building beyond pre-determined thresholds. The application is notified of congestion when it receives a TCAPEVN_CONGEST event that includes the current TCAP congestion level. The congestion level can be between 0 - 3. Zero (0) indicates that congestion has ceased. As the TCAP congestion level increases, the application is expected to reduce its traffic load proportionately until the congestion ceases.

Note: The application receives the TCAPEVN_CONGEST event in the case of either TCAP service congestion or TCAP layer congestion. If it is necessary to distinguish between these two causes, the application can call **TCAPGetApiStats** to check the current congestion level for each of the possible causes to determine the cause of the current congestion.

By default, the TCAP service allocates a buffer pool for up to 128 requests to be queued to the TCAP layer. If the application fails to reduce its traffic load enough to ease the congestion, the TCAP service buffer pool eventually becomes depleted and the TCAP functions fail with a CTAERR_OUT_OF_MEMORY return code. When opening the TCAP service, the application can increase the number of buffers in the pool by setting service argument array element six to a number between 128 and 1024. Increasing the number of buffers in the pool allows a larger burst of traffic to be absorbed without triggering congestion, at the cost of more host memory used. Congestion onset and abatement thresholds are always set to a fixed percentage of the in-use (queued to the TCAP layer) regardless of the total size of the pool, as shown in the following table:

Congestion levels	Onset threshold (to reach this level)	Abatement threshold (to next lower level)
1	Greater than 75% of pool in use	Less than 50% of pool in use
2	Greater than 85% of pool in use	Less than 80% of pool in use
3	Greater than 95% of pool in use	Less than 90% of pool in use

TCAP layer congestion

TCAP layer outbound congestion occurs when the amount of memory available to the TCAP layer for processing new transaction requests falls below configurable thresholds. For the embedded TX board-based TCAP this is the total percentage of available free memory on the board.

Similar to TCAP service congestion, the application is notified of TCAP layer outbound congestion when it receives a TCAPEVN_CONGEST event that includes the current TCAP congestion level. The congestion level can be between 0 - 3. Zero (0) indicates that congestion has ceased. As the TCAP congestion level increases, the application is expected to reduce its traffic load proportionately until the congestion ceases. The TCAP layer also takes action to protect the integrity of the SS7 stack and minimize the impact of the congestion on other services. The following table shows the TCAP layer reaction to outbound congestion:

Congestion level	TCAP action
1	Alarm only - no traffic restriction.
2	No new outbound transactions (BEGIN or QUERY type) allowed. NMS TCAP responds to each new transaction request from an application with a TCAP_STATUS_IND event with a status of TCAP_CONGESTED (unless inbound congestion level is also at level three) and the SAP outbound congestion abort counter is pegged. Other transaction messages (CONTINUE/CONVERSATION, RESPONSE/END, and ABORT) are allowed.
3	All outbound messages are discarded and the SAP outbound congestion discard counter is pegged.

Note: Applications are notified of a change in the outbound congestion level with a TCAPEVN_CONGEST event, and an alarm is generated.

The congestion onset thresholds for TCAP layer congestion can be adjusted with **TCAPGenCfg** or with the *tcapcfg* utility TCMEM_THRESH_X parameters. The congestion abatement thresholds for each congestion level are set midway between the previous two congestion onset thresholds. Refer to the *NMS SS7 Configuration Manual* for information.

Inbound congestion

TCAP inbound congestion is caused by one of two possible conditions:

- The TCAP application is not reading incoming messages as fast as they are generated by the network, resulting in build-up of the user queue.
- The overall memory available to the TCAP layer drops below pre-determined thresholds.

The inbound congestion level of each TCAP user SAP at any given point in time is defined to be the more severe of the current TCAP layer memory congestion level and the user SAP queue congestion level. A low memory condition affects the inbound and outbound congestion level of all user SAPs. A queue build-up congestion condition affects the inbound congestion level of only the user SAP whose queue is building or receding.

Unlike outbound congestion, the TCAP application is not notified directly of inbound congestion level changes, to prevent escalation of the congestion condition. An alarm is generated when a change in a TCAP user SAP's inbound congestion level occurs. The current inbound congestion level of a user SAP can also be determined by calling **TCAPSapStats** or by using the *tcapmgr* utility STATS command.

For inbound congestion, the TCAP layer cannot rely on the application to reduce its traffic load to ease the congestion, as the source of the traffic bursts is generally other network nodes. Instead, the TCAP layer acts directly to control inbound congestion by restricting the types of traffic that are allowed at the various congestion levels. These actions are described in the following table:

Congestion level	TCAP actions
1	Alarm only. There is no traffic restriction.
2	No new inbound transactions are allowed. TCAP responds to each incoming BEGIN (ITU) or QUERY (ANSI) transaction with a P-Abort with a cause of resource limitation (ITU) or resource unavailable (ANSI). The SAP's inbound congestion abort counter is pegged. Other transaction messages are allowed.
3	All inbound messages are discarded and the SAP's inbound congestion discard counter is pegged.

The memory onset and abatement thresholds can be adjusted with the same configuration parameters described in *TCAP service congestion* on page 28.

By default, the user SAP queue congestion thresholds are set to absorb a short burst of approximately 600 messages without the application retrieving a message before congestion control is triggered.

TCAP configuration

NMS SS7 provides a standard configuration program to read the SS7 (including TCAP) configuration from a set of text files and download the configuration to the SS7 tasks running on the TX board. Refer to the *NMS SS7 Configuration Manual* for information. The configuration program (*tcapcfg*) is distributed in both source and executable form. The executable can be used as a standalone configuration tool. The source can be used as a guide for developing a configuration utility using TCAP management functions.

The TCAP layer supports the following configuration entities:

Configuration	Description
	The general configuration defines the resource allocation for the TCAP layer: • Maximum number of user SAPs • Maximum number of simultaneous dialogs • Maximum number of outstanding invokes The general configuration is loaded only once at system boot time and must be loaded before any other configuration entities.
User SAPs	One user SAP is defined for each application using the TCAP layer services. A user SAP is associated with a single subsystem number and protocol switch type (ANSI-88, ANSI-92, ITU-88, ITU-92, or ITU-97). The user SAP defines the default timer values for invokes issued on the SAP and identifies the SCCP user SAP to be used. Additional user SAPs can be added later, up to the maximum number specified in the general configuration.

SCCP addressing and routing

All SCCP connectionless data requests and connection establishment requests contain a mandatory called and calling party address. The calling address is optional for the connection request message. These addresses are passed to and from applications by the SccpAddr data structure, which is a C-language structure representation of the actual address passed in the SCCP protocol message.

SCCP addresses can take several forms, containing various combinations of point code, subsystem number, and global title. The combination of the address and routing indicator constructed by applications (or received from the SS7 network) together with the SCCP configuration allow these messages to be routed to the correct destination or local application.

For outgoing messages from applications, the called party address in the unitdata request or connection request message is used to route the message. The routing method is chosen based on the value specified in the routing indicator field of the called party address. The routing methods are:

- Point code and subsystem number (ROUTE_PC_SSN)
- Global title (ROUTE_GLT)

Routing by point code and subsystem number

When ROUTE_PC_SSN is chosen, the message is routed to the destination point code/subsystem number (DPC/SSN) specified in the called party address. The subsystem number must be present.

If the DPC is present, a route must be configured for the point code and the subsystem must be configured for that route unless the default routing configuration option is selected.

If the DPC is absent, the message is routed to the point code associated with the first (and typically only) route in the SCCP configuration file, but that point code is not included in the outgoing message. This option is typically used only on point-to-point SS7 links, such as the link between a mobile switching center (MSC) and base station controller (BSC) in a wireless network, where the destination point code is not needed for routing.

If a global title was present in the called party address, it is copied to the outgoing message but the routing indicator remains ROUTE_PC_SSN.

Routing by global title

When ROUTE_GLT is chosen, an address translation must be configured that, when combined with the address mask configured for the application SAP, matches the global title specified in the called party address. The message is routed to the point code and subsystem number from the configured address translation entry. If no subsystem number is configured for that address translation but one is supplied by the application, it is copied to the outgoing message.

If the global title must be further translated at another node, the routing indicator configured in the address translation entry specifies ROUTE_GLT and the point code in the address translation entry specifies the next translator node. If this is the final translation, the address translation entry specifies ROUTE_PC_SSN and the DPC in the address translation entry specifies the final destination point code.

When a global title is specified by the user, the global title is translated (if possible) by the SCCP task, and the global title is included in the outgoing SCCP address along with up to four parameters.

The glTransType, encoding, numPlan, and natAddrInd fields are only used with a global title. Based on the value of the glTitleInd field, some or all of these values are included with a global title in an outgoing SCCP address.

If swType is set to SW_ANSI, two combinations can be included with a global title. The glTitleInd field determines which is selected:

```
global title + translation type (glTitleInd = GLT_TT)
```

```
global title + translation type + numbering plan + encoding (GLT_TT_NP_E)
```

If swType is set to SW_ITU, four combinations can be included with a global title. The glTitleInd field determines which is selected:

```
global title + encoding + nature of address
(glTitleInd = GLT_ITU_FMT1)
```

```
global title + translation type (GLT_ITU_FMT2)
```

```
global title + translation type + numbering plan + encoding (GLT_ITU_FMT3)
```

```
global title + translation type + numbering plan + encoding + nature of address
(GLT_ITU_FMT4)
```

If the user wants the global title to be passed to another node for translation, the ROUTING_IND field can be defined in one of the ADDR sections in the SCCP configuration file. If ROUTING_IND is set to GLT, the global title is passed along with its routing field set to route by global title. The point code field is included, but the subsystem field is not included in the outgoing message.

Refer to *Global title translation* on page 33 for more information.

SCCP address override

When a TCAP begin message is received, its corresponding SCCP called and calling addresses are saved in the transaction context. When the application sends a continue or end message, the application-specified SCCP called address is ignored and the calling address from the received begin message is used instead. To use the application-specified called address on a continue or end message, specify the following field in the User SAP section of the TCAP configuration file:

```
SCCP_ADDR_OVERRIDE      1
```

Global title translation

A global title translation (GTT) translates an array of phone numbers into an associated point code and subsystem for routing to another machine.

This topic provides the following information:

- Setting function parameters
- Setting the SCCP configuration
- Setting variations in global title translation

Setting function parameters

The following example is from a TCAP sample program that uses a global title. This example shows the setting of only the called address SCCP address structure:

```
tInfo.cdAddr.presInd = PRESENT;
tInfo.cdAddr.swType  = SW_ANSI;
tInfo.cdAddr.subsystemInd = SUBSYS_NONE; /* A subsystem does not need to be defined   */
tInfo.cdAddr.pointCodeInd = PTCODE_NONE; /* A point code does not need to be defined  */
tInfo.cdAddr.glTitleInd =  GLT_TT;       /* A global title format MUST be selected    */
tInfo.cdAddr.routingInd =  ROUTE_GLT;    /* The routing flag MUST be set to ROUTE_GLT */
tInfo.cdAddr.natIntInd =   ADDRIND_INT;
//tInfo.cdAddr.subsystem =   destssn;    /* The subsystem is not required             */
//tInfo.cdAddr.pointCode =   pointCode;  /* The point code is not required            */
tInfo.cdAddr.glTransType = 1;            /* The global title fields MUST be filled    */
tInfo.cdAddr.encoding = ENC_BCD_EVEN;
tInfo.cdAddr.numPlan =  NP_ISDN;
tInfo.cdAddr.natAddrInd = NATIND_NATL;
tInfo.cdAddr.glTitleLen = 5;      /* The global title length is the BCD-encoded length*/
/* BCD encode the phone number */
for ( i = 0; i < 5; i++)
{
ch = gtitle[((i*2)+1)] - 0x30;
tInfo.cdAddr.glTitle[i] = (ch << 4) & 0xF0;
ch = gtitle[(i*2)] - 0x30;
tInfo.cdAddr.glTitle[i] += ch;
}
```

The global title must be BCD encoded. The global title 8471234567 is BCD encoded as:

```
tInfo.cdAddr.glTitle[0] = 0x48;
tInfo.cdAddr.glTitle[1] = 0x17;
tInfo.cdAddr.glTitle[2] = 0x32;
tInfo.cdAddr.glTitle[3] = 0x54;
tInfo.cdAddr.glTitle[4] = 0x76;
```

The original global title was 10 digits long. When the global title is BCD encoded, it is five bytes in length. This length is used as the global title length.

```
tInfo.cdAddr.glTitleLen = 5;
```

Setting the SCCP configuration

Once the application sends the TCAP message with the route by global title flag set, the SCCP task receives the message and attempts the global title translation. The SCCP task uses three steps to translate a global title:

Step	Description
1	The SCCP task masks the outgoing global title with the ADDR_MASK field in the USER SAP section of the SCCP configuration file. In the previous example, the global title was 8471234567. An ADDR_MASK of FFF masks the first three digits of the global title. The result of the masking in the example is the first three digits - 847. On incoming messages, the ADDR_MASK in the NSAP section of the SCCP configuration file masks global titles.

2	The SCCP task matches the masked result with the ADDRESS sections in the SCCP configuration file. If a matching section is not found, the SCCP message is returned to the application.
	In the example, an ADDRESS 847 section is configured in the SCCP configuration file and matches the masked global title:
	`ADDRESS 847 # global title matching characters`
	`REPLACE_GLT FALSE # do NOT replace the global title`
	`SWITCH_TYPE ANSI # one of ITU, ANSI`
	`NI_IND NATIONAL # one of NATIONAL, INTERNATIONAL`
	`DPC 1.1.2 # translated destination point code`
	`SSN 254`
	`ROUTING_IND PC_SSN # set outgoing routing flag(PC_SSN or GLT)`
	`END #`

3	The SCCP task modifies the SCCP called address of the outgoing message. In the example, the outgoing message has a called address of:
	`cdAddr.pointCode = 1.1.2`
	`cdAddr.subsystem = 254`
	`cdAddr.routingInd = "Route by PC_SSN"`
	The global title specified by the application is also carried in the outgoing message.
	Note: The point code used in the translated message must be listed as a ROUTE in the SCCP configuration file.

Setting variations in global title translation

The following table describes the most common variations to global title translation:

Variation	Description

Forward a global title translation to another node	The following example shows a global title translation block (ADDRESS section) in the SCCP configuration file. 1.1.2 is the point code of the node that performs the global title translation:
	`ADDRESS 847 # global title matching characters`
	`REPLACE_GLT FALSE # do NOT replace the global title`
	`SWITCH_TYPE ANSI # one of ITU, ANSI`
	`NI_IND NATIONAL # one of NATIONAL, INTERNATIONAL`
	`DPC 1.1.2 # forward translation to this point code`
	`ROUTING_IND GLT # set outgoing routing flag (PC_SSN or GLT)`
	`END #`
	This translation block forwards a message to another node (1.1.2) to do the global title translation.
	The ROUTING_IND field is set to GLT (route by global title), which sets the routing flag in the outgoing message. The DPC field contains the point code of the node that receives the message and translates the global title. The SSN field does not need to be defined. The called address of the outgoing message is set to:
	`cdAddr.pointCode = 1.1.2`
	`cdAddr.subsystem = "not encoded"`
	`cdAddr.routingInd = "Route by Global Title"`
	The global title specified by the application is also carried in the outgoing message.
	Note: The point code of the forwarded node must be listed as a route in the SCCP configuration file.

Configure a single ADDRESS section that matches any global title	Set the ADDR_MASK in the USER SAP section of the SCCP configuration file to 0 (zero). Create an ADDRESS 0 translation block. This block matches all global titles.
Do not encode a point code in the translated outgoing message	Remove the DPC field from the ADDRESS translation block. The called address DPC is not encoded in the outgoing message. Use the default routing option to provide an MTP header DPC.
Copy the application specified subsystem in the translated outgoing message	Remove the SSN field from the ADDRESS translation block. The subsystem field specified by the application in the SCCP called address (if it exists) is inserted in the SCCP called address of the outgoing message.

Status and notify indications

The application is notified of errors encountered in processing transaction requests by the TCAP layer at any time with unsolicited TCAP status indication messages (event type TCAP_EVENT_STA_IND). The status field returned in the status indication message indicates the reason for the error.

Transaction requests that could not be delivered by the SCCP layer due to routing errors, global title translation failures, or temporary network outages, for example, are returned to the originating application if requested through the SCCP quality of service parameters with unsolicited TCAP notify indication messages (event type TCAP_EVENT_NOT_IND). The notify indication includes a return cause field that identifies the reason the message could not be delivered.

Dialog IDs

A TCAP application uses two dialog IDs that are 32-bit unsigned integers to reference a particular transaction:

- Service user dialog ID (suDlgId)
- Service provider dialog ID (spDlgId)

The service user dialog ID is assigned by the application on the first outgoing request for a particular transaction. This ID must be unique among all concurrently active transactions associated with a particular TCAP SAP (subsystem number). The content of this dialog ID is not checked by the TCAP protocol layer, but it is passed back and forth between the application and the TCAP layer on all subsequent requests or indications belonging to that transaction. The application can choose any value for the suDlgId such as an address or index to a data structure, to associate a message with its transaction. A suDlgId value can be reused by the application any time after the previous transaction using that value has completed.

The service provider dialog ID is assigned by the TCAP layer in the first incoming message associated with a particular transaction (either a new transaction indication or a response from a far SP). The application is expected to store this value and pass it back to the TCAP layer on all subsequent requests belonging to that transaction. On the first outgoing request for a transaction, and on any subsequent request prior to receiving an incoming message belonging to that transaction, the application must pass a spDlgId value of zero.

4 Using the TCAP service

Setting up the Natural Access environment

Before calling any TCAP service functions, the application must:

- Initialize Natural Access
- Create queues and contexts
- Bind to the TCAP service

Refer to the *Natural Access Developer's Reference Manual* for information about Natural Access.

Initializing the Natural Access environment

The Natural Access environment is initialized by calling **ctaInitialize**. Initialize Natural Access only once per application, regardless of the number of queues and contexts created.

```
CTA_INIT_PARMS      tcapInitparms       = {0};
CTA_SERVICE_NAME    tcapServiceNames[]  = {{"TCAP", "TCAPMGR"}};
...
tcapInitparms.size              = sizeof(CTA_INIT_PARMS);
tcapInitparms.traceflags        = CTA_TRACE_ENABLE;
tcapInitparms.parmflags         = CTA_PARM_MGMT_SHARED;
tcapInitparms.ctacompatlevel    = CTA_COMPATLEVEL;

Ret = ctaInitialize(tcapServiceNames, 1, &tcapInitparms);
if (Ret != SUCCESS) {
    printf("ERROR code 0x%08x initializing CT Access.", Ret);
    exit( 1 );
}
```

Creating queues and contexts

The application creates the required Natural Access queues and contexts. The queue must always be created before any associated context is created.

```
CTAHD       ctaHd;                  /* CTA context handle */
CTAQUEUEHD  ctaQueue;               /* Queue */
...

Ret = ctaCreateQueue( NULL, 0, &ctaQueue );
if ( Ret != SUCCESS )
{
     ctaGetText( NULL_CTAHD, Ret, sErr, sizeof( sErr ) );
     printf( "*ERROR : ctaCreateQueue failed( %s )\n", sErr );
     ...
}

sprintf( contextName, "TcapSAP-%d", spId ); /* context name is optional */

Ret = ctaCreateContext( ctaQueue, spId, contextName, &ctaHd );
if ( Ret != SUCCESS )
{
     ctaGetText( NULL_CTAHD, Ret, sErr, sizeof( sErr ) );
     printf( "ERROR : ctaCreateContext failed( %s )\n", sErr );
     ctaDestroyQueue( pSap->ctaQueue );
     ...
}
```

Binding to the TCAP service

Once the queues and contexts are created, the application must bind to each desired TCAP user service access point by calling **ctaOpenServices** once for each binding. The binding operation specifies the following parameters:

Parameter	Description
board	TX board number.
srcEnt	Calling application entity ID.
srcInst	Calling application instance ID.
suId	Calling application service user ID.
spId	TCAP service access point ID on which to bind.
ssn	TCAP subsystem number associated with the service access point.
API queue size	Maximum number of requests that can be queued to the board within the TCAP service. Valid range is 128 to 1024. Default = 128.

In Natural Access, these parameters are specified in the CTA_SERVICE_ARGS structure, contained in the CTA_SERVICE_DESC structure. An example of the parameter specification is provided:

```
CTA_SERVICE_DESC TCAPOpenSvcLst[] = {{{"TCAP", "TCAPMGR"}, {0}, {0}, {0}}};

tcapOpenSvcLst[0].svcargs.args[0] = boardNum;        /* board number   */
tcapOpenSvcLst[0].svcargs.args[1] = DPRCHAN + sapid; /* srcEnt         */
tcapOpenSvcLst[0].svcargs.args[2] = ZERO;            /* srcInst        */
tcapOpenSvcLst[0].svcargs.args[3] = SUID;            /* suId           */
tcapOpenSvcLst[0].svcargs.args[4] = sapid;           /* spId           */
tcapOpenSvcLst[0].svcargs.args[5] = ssn;             /* ssn            */
tcapOpenSvcLst[0].svcargs.args[6] = 256;
                                        /* increase API queue size */
```

ctaOpenServices is an asynchronous function. The return from the function indicates that the bind operation initiated. Once completed, a CTAEVN_OPEN_SERVICES_DONE event is returned to the application.

Note: Only a single thread should call **ctaOpenServices** to open the TCAP service for a single board. All messages generated by the TCAP task on a single board are reported to the last thread to call **ctaOpenServices** for the TCAP service.

Receiving TCAP service events

After binding to a TCAP user SAP with **ctaOpenServices**, the application receives TCAP service events by periodically calling **ctaWaitEvent**, specifying the Natural Access queue handle. The TCAP service can generate data events and congestion events.

```
ret = ctaWaitEvent( ctaQueue, &event, CTA_WAIT_FOREVER );
if ( ret != SUCCESS )
   /* handle the error */
else
{
   switch ( event.id )
   {
      case TCAPEVN_DATA:
         /* an TCAP date event has occured, call TCAPRetrieveMessage()
            to retrieve the message and proccess it */
         tcret = TCAPRetrieveMessage( ctaHd, &msg, &infoBlk );
         if( ret == TCAP_SUCCESS )
             /* process received TCAP event */
         else if( ret == TCAP_NOMSG
             /* this is normal - just ignore and wait for next event */
         else
             /* TCAPRetrieveMessage failed - handle the error */

         break;

      case TCAPEVN_CONGEST:
         /* TCAP layer or API is congested or congestion abated;
            take appropriate action */
         cong_lvl = (U8) event.value;
         if( cong_lvl == 0 )
             /* congestion abated - restore traffic to normal levels */
         else
             /* congestion now at level "cong_lvl" - take appropriate
                action to reduce traffic. */

         break;

      .
      .
      .
   }
}
```

When a TCAPEVN_DATA event is received, the application calls **TCAPRetrieveMessage** to retrieve the TCAP message for processing. The message could be a TCAP transaction message, or other type of indication, such as a status or notify event. It is possible for **TCAPRetrieveMessage** to return a value of TCAP_NOMSG, indicating that the event was processed internally by the TCAP service, and there is nothing for the application to process. This usually occurs before a congestion event is generated, but can also happen at other times.

If a TCAPEVN_CONGEST event is received, the outbound congestion level of the TCAP user SAP has changed. The new congestion level is contained in the value field of the event structure returned by **ctaWaitEvent**. Upon receipt of this event, the application should take action to reduce (in the case of congestion onset) or restore (in the case of congestion abatement) the traffic load it is generating.

It is also recommended that the application call **TCAPStateReq** and mark the subsystem in service as soon as it is ready to handle data traffic, in case the subsystem was previously left out of service by an application unbinding from the same TCAP service access point.

Handling redundancy events

After binding to a TCAP user SAP, the application receives a TCAP_EVENT_RUN_STATE event indicating the redundancy state of the TCAP layer on the board. The event type associated with this event indicates one of the following states:

Event type	Description
TCAP_STANDALONE	Application is in a non-redundant configuration. Normal operation can begin.
TCAP_PRIMARY	TCAP task on this board is currently the primary board in a redundant board pair. Normal operation is allowed as long as the board remains the primary.
TCAP_BACKUP	TCAP task on this board is currently the backup board in a redundant board pair, monitoring the status of the primary. No active traffic passes through this SAP until the board becomes the primary member of the pair.
TCAP_BACKUP_READY	Backup task has finished updating its list of open transactions.

The TCAP_EVENT_RUN_STATE event is the first message posted to the application's queue for each SAP after the binding is confirmed. No data traffic (unitdata or connections requests) should be directed to this SAP until this event is received.

Refer to the *SS7 Health Management Developer's Reference Manual* for information on writing redundant TCAP applications.

Generating TCAP transactions

Generating a TCAP transaction requires the following steps:

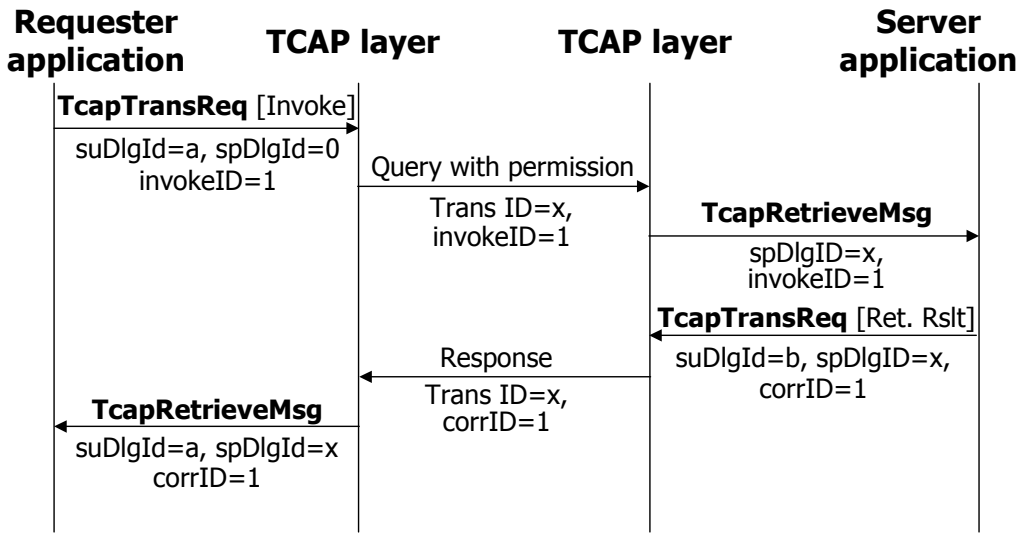
Step	Action
1	Allocate a memory buffer large enough for the transaction request. A TCAP transaction buffer must be at least TCAP_MSG_SIZE bytes.
2	Assign a unique user dialog ID (suDlgId) for the transaction (if it is the first request belonging to this transaction) and populate the transaction information data structure with the: • Message type • Dialog ID(s) • Originating address • Destination address • Quality of service parameters
3	Initialize the transaction request with **TCAPInitTrans**.
4	Add one or more components to the request with **TCAPAddComp**. Each invoke component must also have a unique invoke ID assigned within the current dialog ID.
5	Send the transaction to the destination signaling point by calling **TCAPTransRqst**.

The following examples show how the TCAP service implements a simple request and response transaction and a more complex conversational linked transaction. The ITU-T and ANSI variants are shown separately to clarify the conceptual differences.

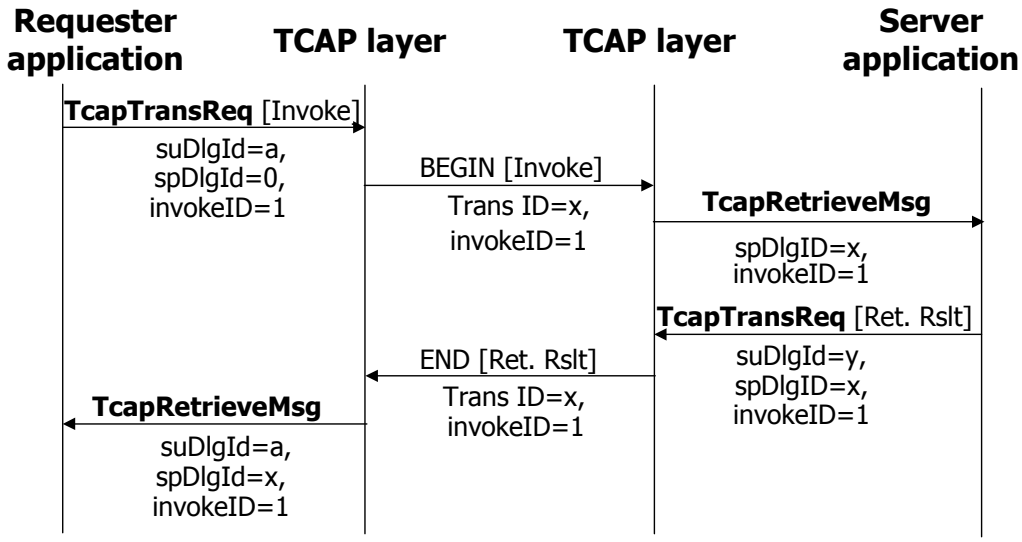
Simple request and response transaction

A simple transaction consists of a transaction begin (ITU-T) or query with permission (ANSI) request with a single invoke component, followed by a single end (ITU-T) or response (ANSI) message with a single return result component. An example transaction is a request from a service switching point (SSP) to a service control point (SCP) to translate an 800 number into a subscriber directory number.

The following illustration shows the ANSI version of a simple transaction:



The following illustration shows the ITU-T version of a simple transaction:



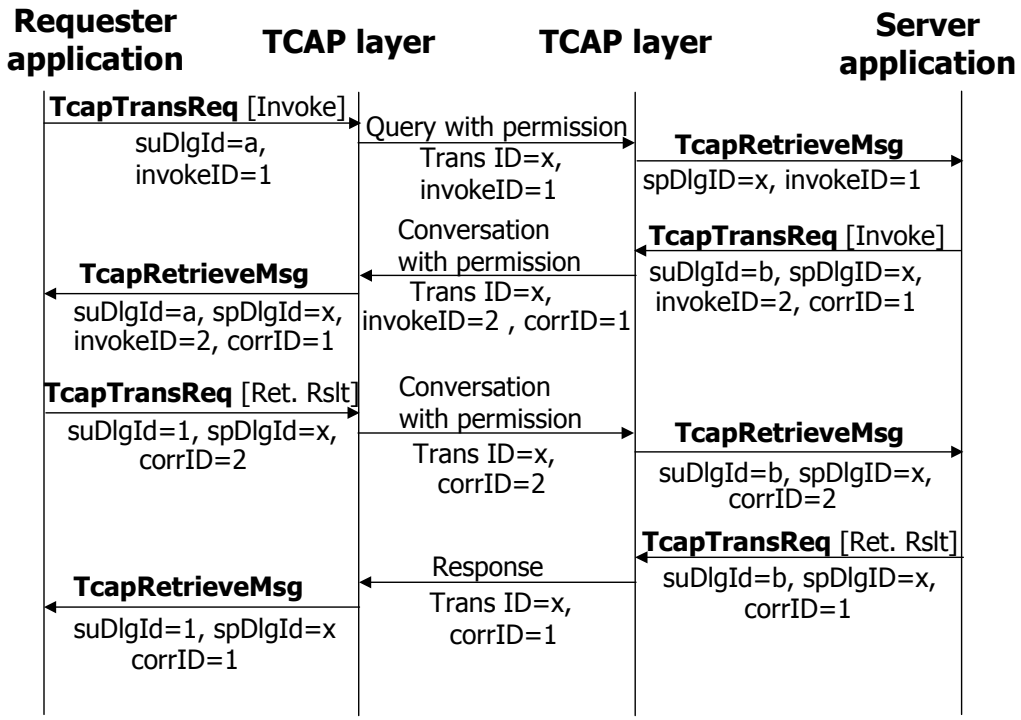
Conversational linked transaction

A complicated conversational and linked transaction can occur as described in the following ANSI and ITU-T examples:

Stage	Description
1	A switching application issues an invoke on behalf of a subscriber to initiate a feature in a remote switch.
2	The remote switch decides it does not have enough information to carry out the feature invocation. It responds with a new invoke (linked to the original invoke) to play an announcement to the subscriber and collect digits.
3	The original switch collects the digits and returns them as the result to invoke #2.
4	The remote switch now completes the feature invocation and returns the result to invoke #1.

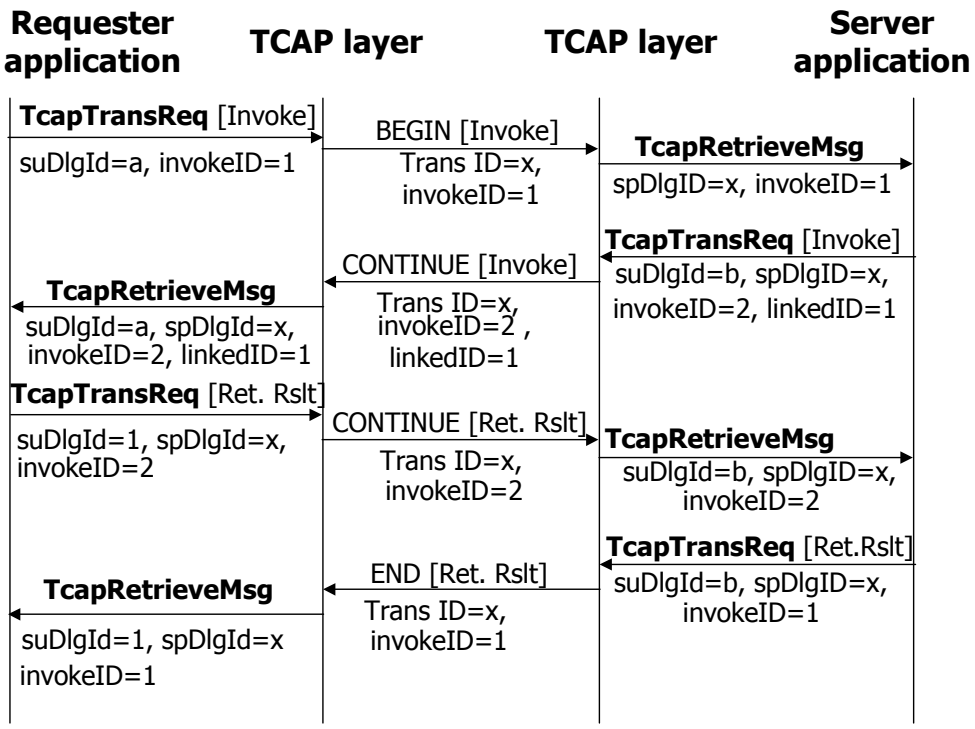
ANSI version

The following illustration shows the ANSI version of a conversational linked transaction:



ITU-T version

The following illustration shows the ITU-T version of a conversational linked transaction:



Handling abnormal conditions

A TCAP message consists of a transaction portion and a component portion. If the transaction portion is invalid, it causes a P-Abort or it is ignored. A P-Abort indication terminates the transaction. An ignored, invalid message does not terminate the transaction. If a TCAP message has an invalid component, a reject component is generated, but the transaction generally is not closed.

The following abnormal condition scenarios are presented:

- Invalid transaction portions
- Transaction inactivity timeouts
- Invalid component in a begin or query message
- Invalid component in a continue or conversation message
- Invalid component in an end or response message
- Invoke time-outs (ITU-T only)
- Invalid component in a multiple component message

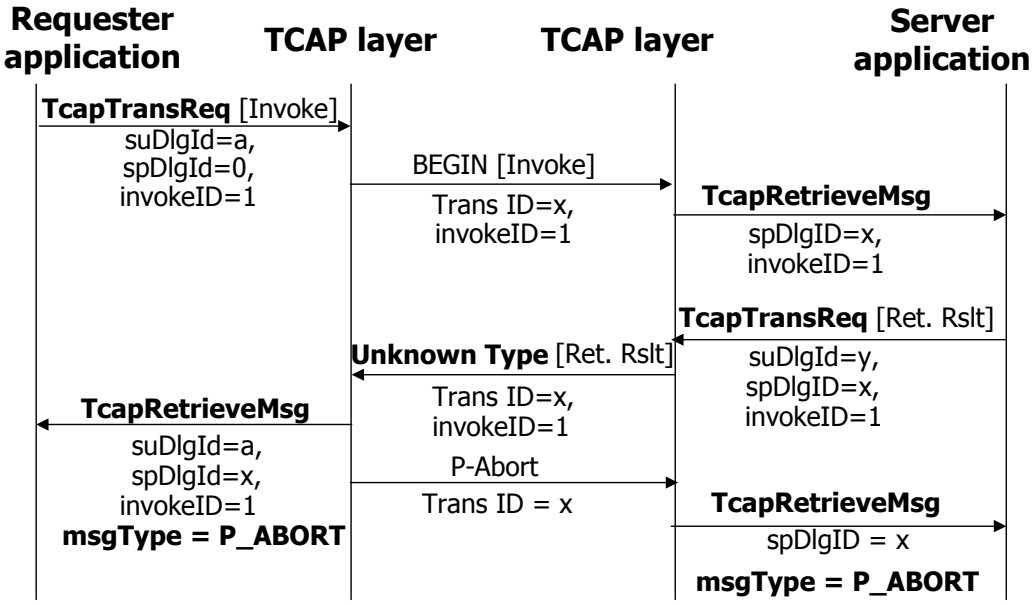
Invalid transaction portions

A TCAP transaction can be received that has an invalid transaction portion. The transaction portion contains the transaction type of begin (ITU-T) or query (ANSI) and any required transaction IDs (TIDs). If an error is detected and a valid transaction ID can be found:

- A P-Abort is returned to the sender.
- A P-Abort indication is delivered to the receiver.
- The transaction is closed.

However, if the TCAP stack cannot find a valid transaction ID, it ignores the message. No response is returned to the sender and no indication is sent to the receiver. As the message is ignored, any transaction that was open remains open.

The following illustration shows a transaction where an ITU-T TCAP message is received that does not have a valid message type of begin, continue, or end. The message does have a valid originating transaction ID. The TCAP stack recognizes the invalid message and sends a P-Abort to the sender. Also, a P-Abort indication is delivered to the receiving application.



Transaction inactivity timeouts

Inactivity timeouts can optionally be specified for TCAP transactions. If a specified period of time elapses with no traffic, an inactivity timeout indication (TCAP_XACTION_TIMEOUT) is sent to the TCAP application. The TCAP application can then:

- Close the transaction through an end, response, or abort message.
- Leave the transaction open with **TCAPRetainTrans**.
- Do nothing and allow the transaction to expire.

TCAPRetainTrans resets the inactivity timer for that transaction. If the inactivity timer expires again and no traffic has occurred, another inactivity timeout indication is sent to the application.

If the application does nothing, and no further traffic occurs for the transaction, the inactivity timer expires a second time, and a P-Abort closes the transaction. The P-Abort indication is sent to the TCAP application, and if traffic has been received, a P-Abort message is sent to the destination.

A default inactivity timer value (INACTIVITY_TIMEOUT) can be specified in each SAP section of the TCAP configuration file. If INACTIVITY_TIMEOUT is set to zero, inactivity timers are not set for any transactions. If INACTIVITY_TIMEOUT is set to a non-zero value, inactivity timers are set to the specified number (in seconds) for every transaction.

A TCAP application can override the default inactivity timer value by setting the inactvTimer field in the TCAPTransInfo structure. If set to zero, the transaction uses the default inactivity timer. If greater than zero, an inactivity timer is set to that number of seconds for that transaction.

Once a transaction has been opened, the inactivity timer value can be modified by sending a Conversation or Continue with a non-zero value in the inactvTimer field in the TcapTransInfo structure. The **TcapRetainTrans** call can also be used to modify the inactivity timer value.

If the inactivity timer is not used, the application must ensure that transactions are closed. If transactions are not closed, the TCAP task eventually runs out of transaction contexts and fails.

A transaction is closed if one of the following conditions occur:

- P-Abort indication is received.
- User abort message is sent or received.
- End (ITU-T) or Response (ANSI) message is sent or received.

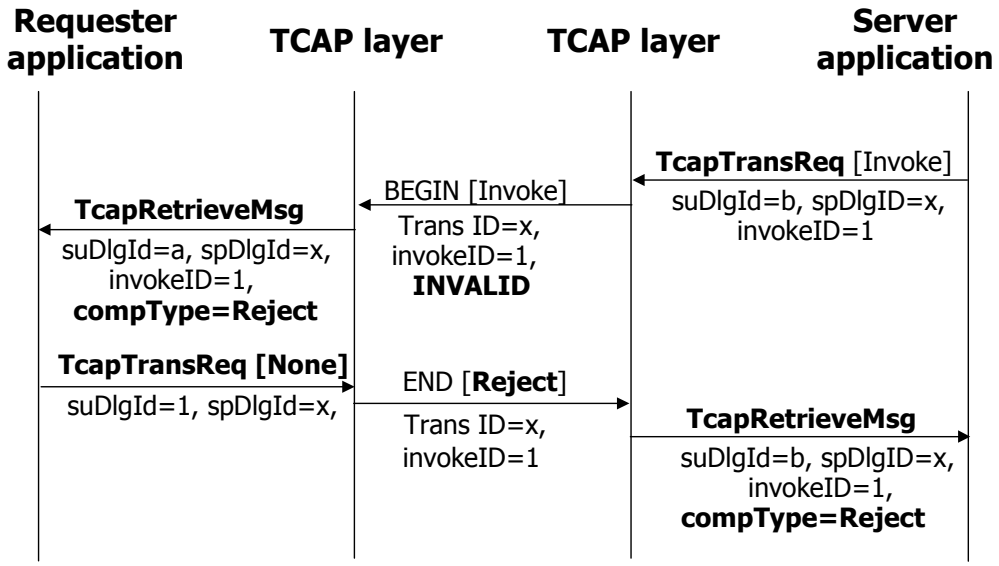
If none of these conditions occur, the application must ensure that the transaction is eventually closed.

Invalid component in a begin or query message

If the TCAP stack receives a begin (ITU_T) or query (ANSI) message with an invalid component, it passes a reject component to the receiving application, but does not automatically send the reject component to the sender. This occurs because the stack is unsure whether to end the transaction, as another component in the message can be valid. In this case, the application must send a continue or an end with no components. The TCAP stack attaches the reject component and sends it to the original sender.

An example of an invalid component is if a return result or return error component is received with an invoke ID that does not correspond to an outstanding invoke component. In this case, a reject component is returned to the sender with the invoke ID it was sent with. If the TCAP stack receives a component with an invalid or missing invoke ID, a reject component is returned to the sender with an invoke ID tag of ASN.1 Universal NULL (0x05). The invoke ID length is set to zero.

In the following illustration, an invalid invoke component is received. The TCAP stack recognizes the error and passes a reject component to the receiving application. Since the reject component is contained in a begin message, the receiving application sends an end message with no components. The TCAP stack attaches the reject component (it has saved the component) and sends it to the original sender. The transaction is now closed.

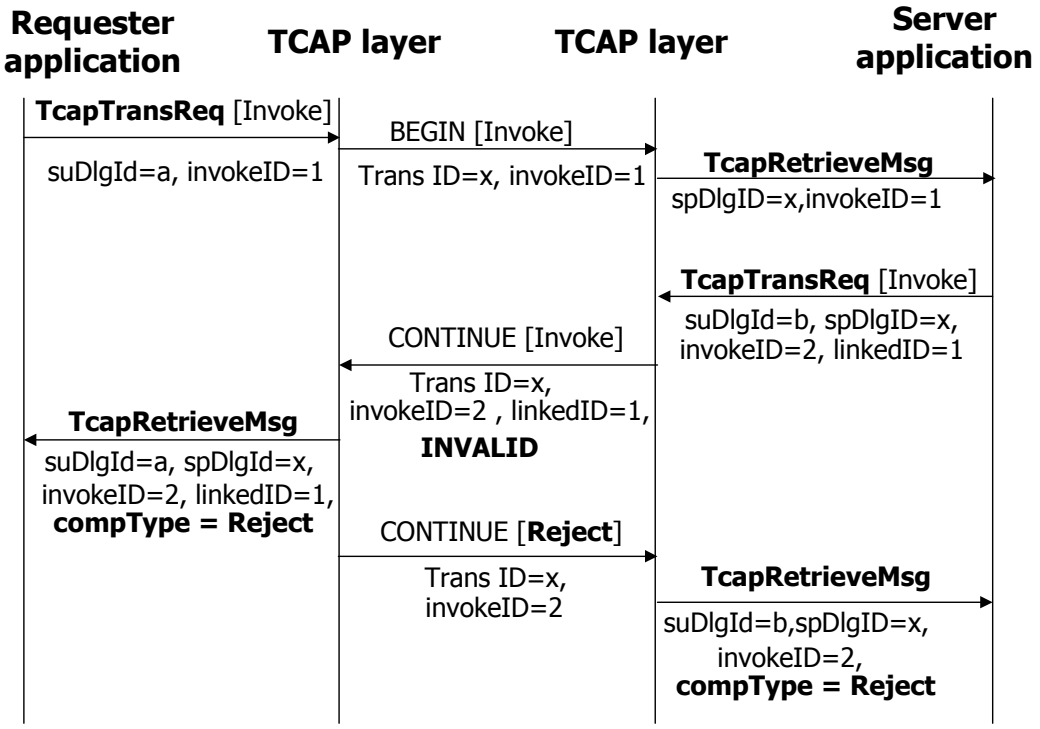


Invalid component in a continue or conversation message

If the TCAP stack receives a continue (ITU-T) or conversation (ANSI) message with a bad component, it automatically sends a continue or conversation with the reject component back to the sender, and sends the reject component to the receiving application. In this case, the application does not need to send a response.

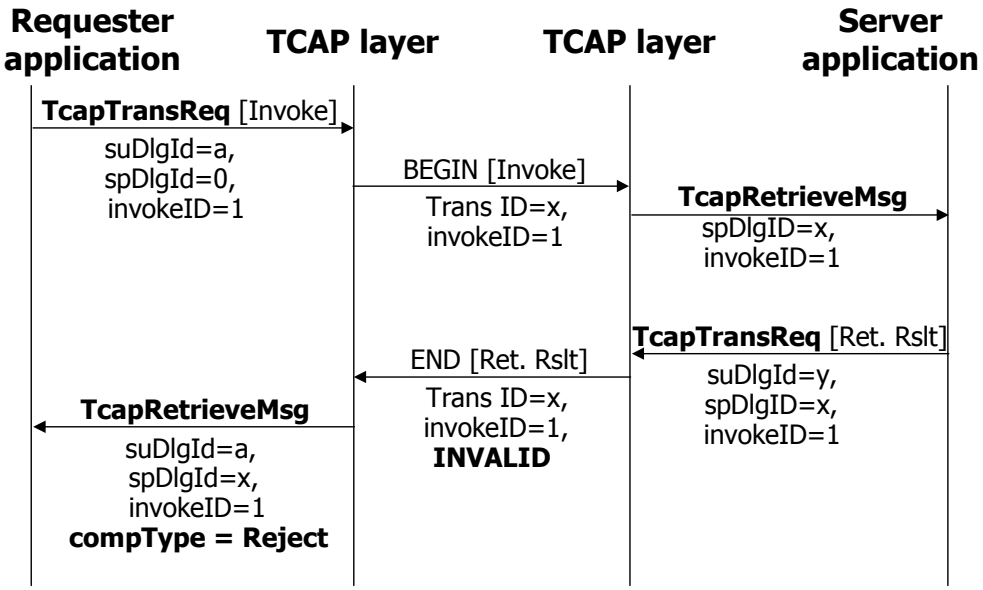
In the following illustration, after sending an invoke component, a continue message with an invalid linked invoke component is received. The TCAP stack automatically sends a continue message with a reject component back to the sender.

In addition, a continue message with a reject component is passed to the receiving application. The transaction is still open and can continue or be closed.



Invalid component in an end or response message

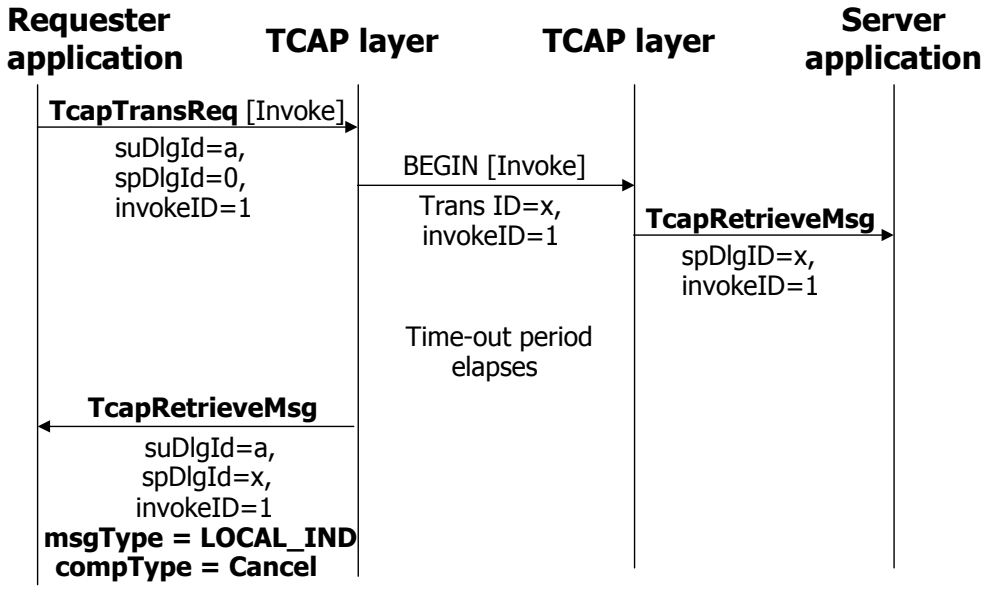
As shown in the following illustration, if the TCAP stack receives an end (ITU-T) or response (ANSI) message with a bad component, it sends the reject up to the receiving application, but nothing back to the sender, as the sender has already closed the transaction. The application does not need to send any response.



Generally, an invoke component is not allowed in an end message. However, invoke components are allowed by setting the ALLOW_INVOKE_END field in the USER_SAP section of the TCAP configuration file. Refer to the *NMS SS7 Configuration Manual* for information.

Invoke time-outs (ITU-T only)

As shown in the following illustration, only the ITU-T protocol implements invoke time-outs. If an application sends an invoke component, and does not receive a return result last, return error, or a reject with the same invoke ID within a configured time, the invoke times-out. A message type of TCAP_LOC_IND is returned with a TCAP_CANCEL component. The application closes the transaction by an end (ITU-T) or a user abort.



Invalid component in a multiple component message

A TCAP message can contain multiple components. As soon as an invalid component is detected by the TCAP stack, any valid components are indicated to the receiving application along with the rejected component. Any components detected after the rejected component are ignored.

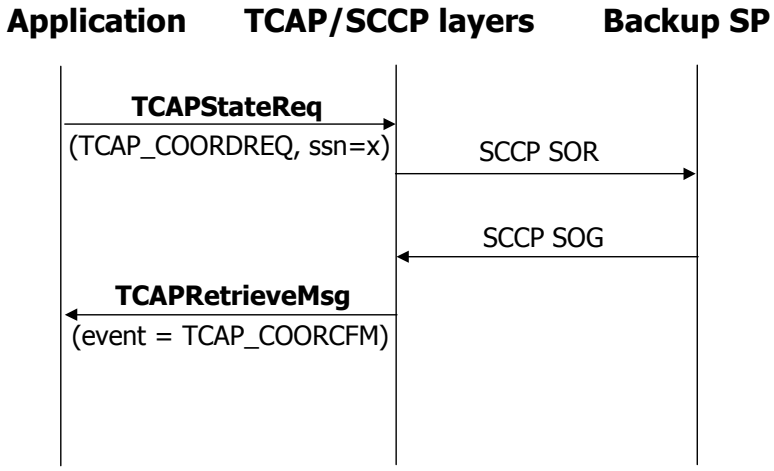
For example, if a TCAP message is received with three components, and the second component is invalid, the first valid component and the second rejected component are indicated to the receiving application. The third valid component is ignored and not indicated to the receiving application.

Signaling point and subsystem status

The TCAP service contains functions that provide TCAP applications with access to the SCCP layer facilities for maintaining signaling point and subsystem status between the calling application's system and backup signaling points or concerned signaling points.

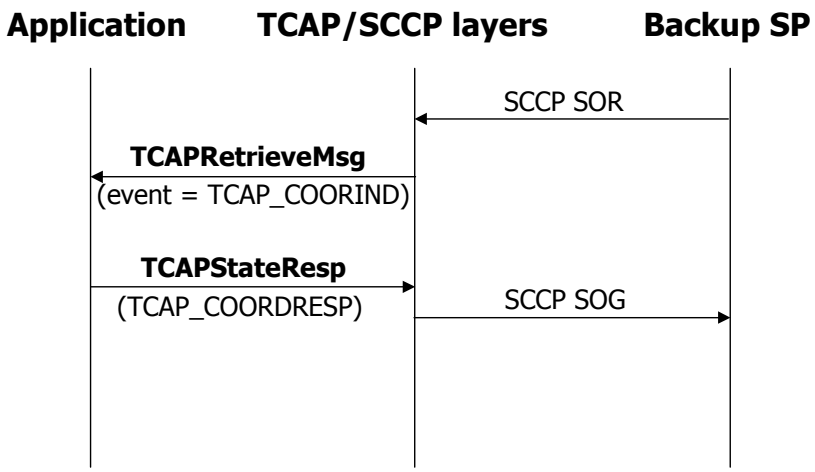
Coordinated state change

An application requests that its subsystem be taken out of service and have all traffic routed to its backup point code by invoking **TCAPStateReq** with a request type of TCAP_COORDREQ. This generates a SCCP subsystem-out-of-service-request (SOR) to the backup signaling point as specified in the SCCP SAP configuration. The application receives an incoming TCAP event with an event type of TCAP_COORDCFM when the backup signaling point returns a subsystem-out-of-service-grant (SOG), as shown in the following illustration:



If the backup signaling point fails to return a SOG message and the grant request times out, the TCAP_COORDCFM event indication contains a value of UOR_DENIED in the subsystem multiplicity indicator (smi) field, implying that the application should not go out of service.

Alternatively, the backup point code requests to go out of service by sending the SOR message. This results in the application receiving a TCAP_COORDIND event as shown in the following illustration. The application invokes **TCAPCoordResp** with an event type of TCAP_COORDRESP to accept the request and return the SOG message.



Subsystem state changes

The application notifies all concerned point codes of a change in its state by invoking **TCAPStateReq** with a status of SS_IS (in service) or SS_OOS (out of service). This request generates a subsystem available (SSA) or subsystem prohibited (SSP) message to all concerned signaling points as specified by the SCCP configuration of the application's SAP.

When the SCCP task receives messages from concerned signaling points indicating that their status has changed, the application receives an unsolicited TCAP event with an event type of TCAP_SSNSTIND (subsystem status) or TCAP_PCSTIND (point code status).

Remote signaling point failures

An application can monitor the status of remote signaling points by specifying a list of concerned point codes in the SCCP user SAP configuration corresponding to that application.

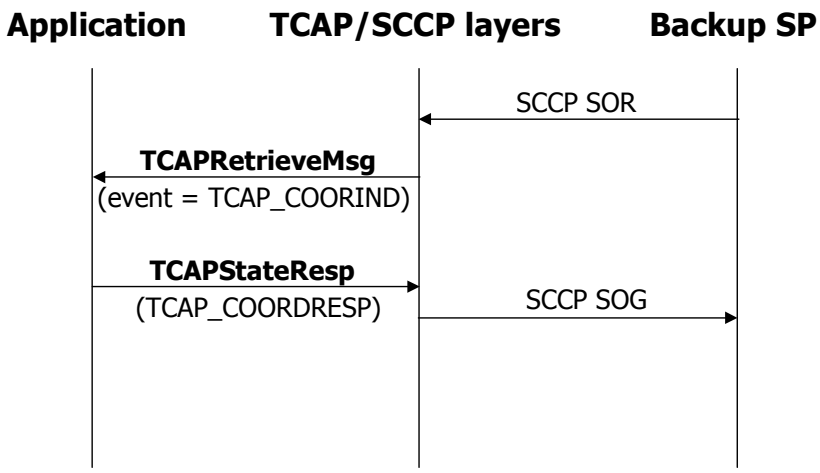
If all routes to a concerned point code (CPC) become unavailable, the application receives an unsolicited TCAP event with an event type of TCAP_PCSTIND with the status field set to SP_INACC (signaling point inaccessible). In addition, the application receives an unsolicited TCAP event with an event type of TCAP_SSNSTIND with the status field set to SS_OOS (subsystem out-of-service) for each known subsystem at that signaling point.

If the MTP layer receives an indication from the remote SP that the SCCP user part is unavailable, the application receives a TCAP_SSNSTIND (SS_OOS) event for each known subsystem at that signaling point. This is not true for the TCAP_PCSTIND event indication, since only the SCCP user part has failed and not the entire signaling point.

When communication with the affected signaling point is restored, the application receives an unsolicited TCAP event with an event type of TCAP_PCSTIND and the status field set to SP_ACC (SP accessible). The SCCP layer initiates subsystem status testing of all known subsystems at the affected SP. When a subsystem available

message is returned by the affected SP, the application receives a TCAP event with an event type of TCAP_SSNSTIND and the status field set to SS_IS (subsystem in-service). The application can re-establish communication with the affected SP/subsystem.

The following example shows a remote signaling point failure and recovery procedure:



Tracing function calls and events

Natural Access provides a mechanism for tracing function calls and events issued or received by an application. To capture trace messages, the Natural Access Server (*ctdaemon*) must be running, and the TCAP service must be included in the [ctasys] section of the *cta.cfg* file, as shown:

```
[ctasys]
Service = tcap, tcapmgr
...
```

In addition, the application must enable tracing when Natural Access is initialized:

```
tcapInitparms.size              = sizeof(CTA_INIT_PARMS);
tcapInitparms.traceflags        = CTA_TRACE_ENABLE;
tcapInitparms.parmflags         = CTA_PARM_MGMT_SHARED;
tcapInitparms.ctacompatlevel    = CTA_COMPATLEVEL;

Ret = ctaInitialize(tcapServiceNames, 1, &tcapInitparms);
if (Ret != SUCCESS) {
    printf("ERROR code 0x%08x initializing CT Access.", Ret);
    exit( 1 );
}
```

For information about tracing, refer to the *Natural Access Developer's Reference Manual.*

5 TCAP service function reference

TCAP service function summary

NMS TCAP consists of the following service functions:

Function	Description
TCAPAddComp	Adds a component and, optionally, its associated parameters to a TCAP transaction data message being constructed.
TCAPCoordReq	Requests that the subsystem be taken out of service and have all traffic routed to its backup point code.
TCAPCoordResp	Accepts a request that an associated signaling point subsystem be taken out of service.
TCAPGetApiStats	Retrieves congestion level activity statistics from the TCAP service.
TCAPGetComp	Extracts a specific component and, optionally, its associated parameters from a TCAP transaction data message received through **TCAPRetrieveMessage**.
	Initializes a new transaction message prior to adding components to the transaction message (with **TCAPAddComp**) or sending the transaction request with **TCAPTransRqst**, or both.
TCAPRetainTrans	Resets the TCAP inactivity timer.
TCAPRetrieveMessage	Retrieves the next message from the TCAP layer.
TCAPStateReq	Notifies all concerned point codes of a change in its subsystem state by generating a subsystem available (SSA) or subsystem prohibited (SSP) message to all concerned signaling points as specified by the application's SCCP service access point.
TCAPTransRqst	Sends a transaction message request to the TCAP layer.

Using the TCAP service function reference

This section provides an alphabetical reference to the TCAP service functions. A prototype of each function is shown with the function description and details of all arguments and return values. A typical function includes:

Prototype	The prototype is followed by a listing of the function arguments. NMS data types include: • U8 (8-bit unsigned) • S16 (16-bit signed) • DWORD (32-bit unsigned) • Bool (8-bit unsigned) If a function argument is a data structure, the complete data structure is shown.
Return values	The return value for a function is either TCAP_SUCCESS or an error code. For asynchronous functions, a return value of TCAP_SUCCESS (zero) indicates the function was initiated; subsequent events indicate the status of the operation.

Use of the TCAP service functions requires the prior installation of the SCCP layer package, particularly the *sccpapi.h* file, which defines data structures and constants required for TCAP applications.

TCAPAddComp

Adds a component and, optionally, its associated parameters to a TCAP transaction data message being constructed.

Prototype

DWORD **TCAPAddComp** (U8 ***pMessage**, TcapComp ***pComp**, U16 ***paramLen**, U8 ***pParamBuf**)

Argument	Description
pMessage	Pointer to the address of the caller's message buffer. This buffer must have been previously initialized with **TCAPInitTrans**.
pComp	

Return values

Return value	Description
TCAP_SUCCESS	
TCAP_OVERFLOW	Adding this component/parameters to the message overflows the maximum TCAP message size.
TCAP_PARAM	NULL pointer was passed for a required parameter.
TCAP_UNINIT	Supplied message buffer (***pMessage***) is not a valid message.

Details

For ANSI components, all parameters associated with a component must be enclosed by an ASN.1 SET or SEQUENCE constructor. This is performed on behalf of the application by the TCAP layer, based on the value in the paramFlg field in the ANSI component structure.

For ITU-T components, there is no requirement that all parameters be enclosed in a ASN.1 SET or SEQUENCE constructor. If this is required by the specific application protocol, then it must be done by the application.

TCAPCoordReq

Requests that the subsystem be taken out of service and have all traffic routed to its backup point code.

Prototype

DWORD **TCAPCoordReq** (CTAHD ***ctahd***, S16 ***spId***, U8 ***aSsn***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext**.
spId	TCAP service access point to which the caller is bound.
aSsn	Affected subsystem number.

Return values

Return value	Description
TCAP_SUCCESS	
CTAERR_BAD_ARGUMENT	Natural Access handle is invalid, or an invalid message buffer has been used.

Details

Generates a SCCP subsystem-out-of-service-request (SOR) to the backup signaling point as specified in the SAP configuration.

TCAPCoordResp

Accepts a request that an associated signaling point subsystem be taken out of service.

Prototype

DWORD **TCAPCoordResp** (CTAHD ***ctahd***, S16 ***spId***, U8 ***aSsn***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext**.
spId	TCAP service access point to which the caller is bound.
aSsn	Affected subsystem number.

Return values

Return value	Description
TCAP_SUCCESS	
	Natural Access handle is invalid, or an invalid message buffer has been used.
CTAERR_DRIVER_SEND_FAILED	Message buffer was not sent to the board. Call **TCAPGetApiStats** and check the txLastErr field to find the error.
CTAERR_INVALID_HANDLE	Natural Access handle is invalid.
CTAERR_OUT_OF_MEMORY	TCAP service queue is full and another request could not be queued to the TCAP layer.

Details

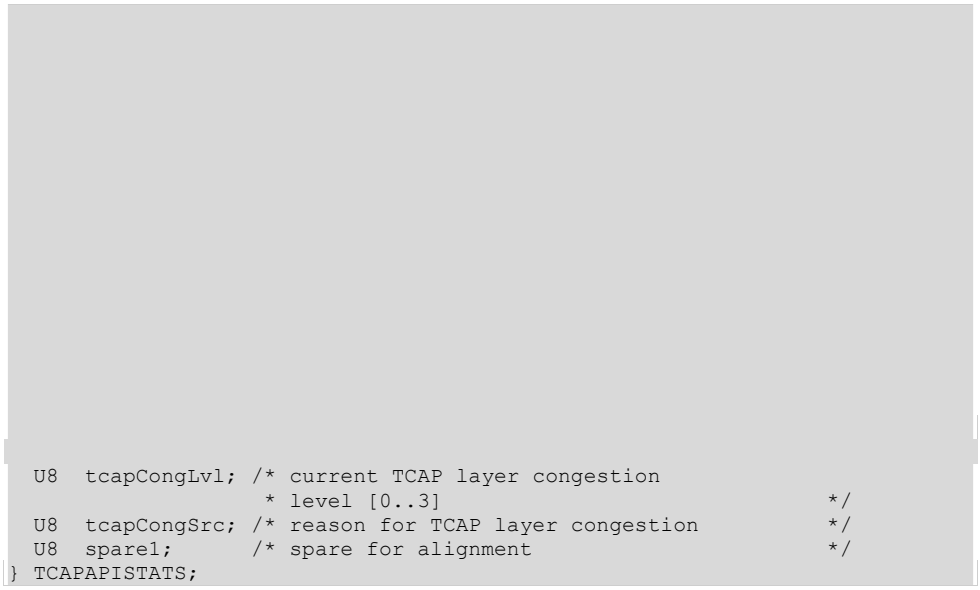
Generates a SCCP subsystem-out-of-service-grant (SOG) to the backup signaling point as specified in the SAP configuration.

TCAPGetApiStats

Retrieves congestion level activity statistics from the TCAP service. For more information, refer to *TCAP service congestion* on page 28.

Prototype

DWORD **TCAPGetApiStats** (CTAHD ***ctahd***, TCAPAPISTATS ****pStats***, BOOL ***reset***)

Argument	Description
ctahd	
	
	If set to TRUE, all statistics (but not current congestion level) are reset to zero after statistics are returned.

Return values

Return value	Description
TCAP_SUCCESS	
CTAERR_BAD_ARGUMENT	Natural Access handle is invalid, or the statistics buffer is set to NULL.
CTAERR_INVALID_HANDLE	Natural Access handle is invalid.

TCAPGetComp

Extracts a specific component and, optionally, its associated parameters from a TCAP transaction data message received through **TCAPRetrieveMessage**.

Prototype

DWORD **TCAPGetComp** (U8 ***pMessage***, U8 ***index***, TcapComp ***pComp***, U16 ***pParamLen***, U8 ***pParamBuf***)

Argument	Description
pMessage	Pointer to the address of the caller's message buffer containing a message of type TCAP_EVENT_DAT_IND that was previously received through **TCAPRetrieveMessage**.
index	Index of the component being retrieved. Valid range is 0 to (TcapTransEvent.numComps - 1), where TcapTransEvent is the event structure associated with this message that was returned by **TCAPRetrieveMessage**.
pComp	Pointer to the address of the caller's TcapComp area where the component body is returned.
pParamLen	On input, this field is set to the maximum number of bytes of parameters to be copied into the caller's parameter buffer. On return, this field is set to the actual number of bytes of parameters copied to the caller's parameter buffer.
pParamBuf	Pointer to the address of the caller's parameter buffer where parameters belonging to this component are returned to the caller. The size of this buffer is passed initially in ***pParamLen***.

Return values

Return value	Description
TCAP_SUCCESS	
TCAP_INDEX	Specified component index does not exist in the message buffer passed by the caller, or it was not a valid TCAP transaction message in the buffer.
TCAP_OVERFLOW	Caller's parameter buffer was not large enough to hold the parameters for this component.
TCAP_PARAM	***pComp*** buffer is NULL.
TCAP_UNINIT	Supplied message buffer (***pMessage***) is not a valid message.

Details

Preserve the contents of the message buffer between the call to **TCAPRetrieveMessage** that returned the message and the last call to **TCAPGetComp** to retrieve the last component. Once the last call to **TCAPGetComp** returns, the message buffer can be released or reused.

If the caller does not expect any parameters with this component, then NULL pointers can be specified for ***pParamLen*** and ***pParamBuf***.

If the number of bytes of component parameters belonging to the specified component exceeds the size of the caller's parameter buffer (as specified by the input value of ***pParamLen***), then:

- Only the first (***pParamLen***) bytes of parameters are copied to the caller's parameter buffer (if ***pParamLen*** and ***pParamBuf*** are not NULL).
- The actual number of bytes needed to contain the parameters is returned to the caller in ***pParamLen*** (if ***pParamLen*** is not NULL).
- The return value of the function is TCAP_OVERFLOW.

TCAPInitTrans

Initializes a new transaction message prior to adding components to the transaction message (with **TCAPAddComp**) and/or sending the transaction request with **TCAPTransRqst**.

Prototype

DWORD **TCAPInitTrans** (U8 ***pMessage***, TcapTransInfo ***pTinfo***, TcapDlgSect ***pDlgPart***, U16 ***userInfoLen***, U8 ***pUserInfo***)

Argument	Description
pMessage	Pointer to the address of the caller's message buffer, where the TCAP message is constructed. This buffer must be at least TCAP_MSG_SIZE bytes long (currently 1076).
pTInfo	Pointer to the address of the caller's TcapTransInfo buffer specifying the general characteristics of the transaction message.
pDlgPart	Pointer to the address of the caller's TcapDlgSect structure (optional).
pUserInfoLen	Byte length of user data to be included in the message (optional).
pUserInfo	Pointer to the address of the caller's user data to be included in the message.

Return values

Return value	Description
TCAP_SUCCESS	
TCAP_OVERFLOW	Caller's user information is too big for a TCAP message.
TCAP_PARAM	NULL pointer was passed for a required parameter.

Details

After successful completion of **TCAPInitTrans**, the caller can then call **TCAPAddComp** one or more times to add components to the transaction message. The application must then call **TCAPTransRqst** to send the constructed message to the TCAP layer.

For information about the structure of the transaction information block that is pointed to in ***pTinfo***, refer to *TCAP transaction information structure* on page 101.

The dialog portion is valid only for ITU-92 or later and ANSI-96 or later protocols. If no dialog portion is in the outgoing message, a NULL pointer should be passed in ***pDlgPart***. For information about the structure of the TcapDlgSect block, refer to *TCAP dialog section structure* on page 103.

The user information portion is valid only for ITU-92 or later and ANSI-96 or later protocols. For other protocol variants, or if no user data is included, a NULL pointer should be passed in ***pUserInfo*** and ***pUserInfoLen*** should be set to zero.

The application has 912 bytes in the message buffer to specify the dialog section, user information section, and components. If a dialog section is specified, about 600 bytes remain for the user information section, and the components and their parameters. Each component uses 88 bytes plus the size of its parameter.

TCAPRetainTrans

Resets the TCAP inactivity timer.

Prototype

DWORD **TCAPRetainTrans** (CTAHD ***ctahd***, S16 ***spId***, TcapTransInfo ****pTinfo***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext**.
spId	TCAP service access point ID.
pTinfo	Pointer to the address of the caller's TcapTransInfo buffer specifying the general characteristics of the transaction message.

Return values

Return value	Description
TCAP_SUCCESS	
CTAERR_BAD_ARGUMENT	Natural Access handle is invalid, or an invalid message buffer has been used.
CTAERR_DRIVER_SEND_FAILED	Message buffer was not sent to the board. Call **TCAPGetApiStats** and check the txLastErr field to find the error.
CTAERR_INVALID_HANDLE	Natural Access handle is invalid.
CTAERR_OUT_OF_MEMORY	TCAP service queue is full and another request could not be queued to the TCAP layer.

Details

This function is normally called after an inactivity timeout indication was received for an open transaction. If the application wants to keep the transaction open, a call to **TCAPRetainTrans** causes the inactivity timer to be reset. When the inactivity timer expires again, another inactivity timeout indication is received.

The inactivity timer value can be modified by setting a non-zero value in the inactvTimer field in the TcapTransInfo structure. If set to zero, the inactivity timer value remains the same.

TCAPRetrieveMessage

Retrieves the next message from the TCAP layer.

Prototype

DWORD **TCAPRetrieveMessage** (CTAHD ***ctahd***, U8 ****pMessage***, TcapRecvInfo ****pInfoBlk***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext**.
pMessage	Pointer to the address of the caller's message buffer where the received message is returned to the caller. This buffer must be at least TCAP_MSG_SIZE bytes long (currently 512).
pInfoBlk	Pointer to the address of the caller's receive information block where information regarding the received message is returned to the caller.

Return values

Return value	Description
TCAP_SUCCESS	
TCAP_DRIVER	Error occurred accessing the CPI driver.
TCAP_NOMSG	No event messages are waiting.

Details

Periodically call this function to receive messages from the TCAP layer.

When a message is received, **TCAPRetrieveMessage** copies the message to the caller's message buffer and performs any necessary byte order translation to convert to the host's native byte ordering. Information about the received message is returned to the caller in the ***pinfoBlk*** parameter. The event structure associated with a received message and the information returned in the receive information block depend on the type of message received from the TCAP layer (as determined by the value of the ***pinfoBlk.indType*** field).

For all events other than TCAP_EVENT_DAT_IND events, all relevant event information is copied from the message buffer directly to the caller's receive information block and the message buffer can be deallocated or reused immediately upon return from this function.

For TCAP_EVENT_DAT_IND events, however, the component portion, application component, parameters, and any user dialog information present in the message remain in the caller's message buffer upon return from this function. Therefore, the caller must preserve the contents of the message buffer until all component data, component parameters, and dialog portion user information has been retrieved through **TCAPGetComp** and processed completely or copied to a safe location.

pinfoBlk defines the type of event received and other event-specific attributes of the incoming message. For more information, refer to *General receive information block structure* on page 115.

TCAPStateReq

Notifies all concerned point codes of a change in its subsystem state by generating a subsystem available (SSA) or subsystem prohibited (SSP) message to all concerned signaling points as specified by the application's SCCP service access point.

Prototype

DWORD **TCAPStateReq** (CTAHD ***ctahd***, S16 ***spId***, U8 ***aSsn***, U8 ***status***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext**.
spId	TCAP SAP ID to which the caller is bound.
aSsn	Affected subsystem number.
status	New subsystem status: 0x03 = SS_OOS Subsystem out of service 0x04 = SS_IS Subsystem in service

Return values

Return value	Description
TCAP_SUCCESS	
CTAERR_BAD_ARGUMENT	Natural Access handle is invalid, or an invalid message buffer has been used.
CTAERR_DRIVER_SEND_FAILED	Message buffer was not sent to the board. Call **TCAPGetApiStats** and check the txLastErr field to find the error.
CTAERR_INVALID_HANDLE	Natural Access handle is invalid.
CTAERR_OUT_OF_MEMORY	TCAP service queue is full and another request could not be queued to the TCAP layer.

TCAPTransRqst

Sends a transaction message request to the TCAP layer.

Prototype

DWORD **TCAPTransRqst** (CTAHD ***ctahd***, S16 ***spId***, U8 ****pMessage***)

Argument	Description
ctahd	Context handle returned by **ctaCreateContext**.
spId	TCAP SAP ID to which this transaction belongs.
pMessage	Pointer to the address of the caller's transaction message that was constructed with **TCAPInitTrans**, and optionally, **TCAPAddComp**.

Return values

Return value	Description
TCAP_SUCCESS	
TCAP_PARAM	Message buffer is not valid. Call **TCAPInitTrans** using the message buffer before you call **TCAPTransRqst**.
CTAERR_BAD_ARGUMENT	Natural Access handle is invalid, or an invalid message buffer has been used.
CTAERR_BAD_SIZE	Message buffer is too large to be sent.
CTAERR_DRIVER_SEND_FAILED	Message buffer was not sent to the board. Call **TCAPGetApiStats** and check the txLastErr field to find the error.
CTAERR_INVALID_HANDLE	Natural Access handle is invalid.
CTAERR_OUT_OF_MEMORY	TCAP service queue is full and another request could not be queued to the TCAP layer. Refer to *TCAP service congestion* on page 28 for more information.

Details

Successful completion of this function implies that the request was successfully queued to the TCAP layer. The request can subsequently be rejected by the TCAP layer. If the request is rejected, the application receives a status event indication (TCAP_EVENT_STA_IND) message from the TCAP layer. If the request cannot be delivered by the SCCP layer, the application receives a notify event indication (TCAP_EVENT_NOT_IND) message from the TCAP layer.

The transaction message must have been constructed using **TCAPInitTrans**, and optionally **TCAPAddComp**, prior to being sent.

6 TCAP management function reference

TCAP management function summary

NMS TCAP consists of the following management functions:

Function	Description
TCAPAlarmControl	Controls the level of alarms generated by the TCAP task on the TX board.
TCAPGenCfg	Sends the TCAP general configuration parameters to the TX board.
TCAPGenStatus	Retrieves the TCAP general status structure.
TCAPGetGenCfg	Retrieves the current values for the general configuration parameters from the TX board.
TCAPGetSapCfg	Retrieves the current configuration parameter values for a specific TCAP SAP.
TCAPInitGenCfg	Initializes a TCAP general configuration buffer to default configuration values that can be passed to **TCAPGenCfg**.
TCAPInitMgmtAPI	Initializes TCAP management and provides access to the TX board.
TCAPInitSapCfg	Builds a default TCAP user SAP configuration buffer that can be passed to **TCAPSapCfg**.
TCAPSapCfg	Sends a TCAP service access point (SAP) configuration parameter block to the specified TX board to define or update the configuration for a specific TCAP SAP.
TCAPSapStats	Retrieves and optionally resets the statistics for a specified TCAP service access point (SAP).
TCAPTermMgmtAPI	Terminates the connection between the application and the TX board, releasing any resources (file descriptors) associated with TCAP management functions.
TCAPTraceControl	Sends a request to enable or disable tracing of TCAP protocol messages.

Using the TCAP management function reference

This section provides an alphabetical reference to the TCAP management functions. A prototype of each function is shown with the function description and details of all arguments and return values. A typical function includes:

Prototype	The prototype is followed by a listing of the function arguments. NMS data types include: • U8 (8-bit unsigned) • S16 (16-bit signed) • U16 (16-bit unsigned) • U32 (32-bit unsigned) • Bool (8-bit unsigned) If a function argument is a data structure, the complete data structure is shown.
Return values	The return value for a function is either TCAP_SUCCESS or an error code. For asynchronous functions, a return value of TCAP_SUCCESS (zero) indicates the function was initiated; subsequent events indicate the status of the operation.

Unlike the TCAP service functions that send and receive messages asynchronously, each TCAP management function generates a request followed immediately by a response from the TX board. TCAP management functions block the calling application waiting for this response (typically a few hundred milliseconds) and return an indication as to whether or not an action was completed successfully. For this reason, the TCAP management functions are typically used by one or more management applications, separate from the applications that use the TCAP service functions. TCAP management is packaged as a separate library with its own interface header files.

TCAPAlarmControl

Controls the level of alarms generated by the TCAP task on the TX board.

Prototype

S16 **TCAPAlarmControl** (U8 ***board***, U8 ***alarmLvl***)

Argument	Description
board	TX board number to which this request is directed.
alarmLvl	New alarm level.

Return values

Return value	Description
TCAP_SUCCESS	
TCAPM_BOARD	***board*** is out of range.
TCAPM_DRIVER	Error occurred accessing the driver.
TCAPM_FAILED	Task on the TX board reported a failure.
TCAPM_TIMEOUT	Request timed out.
TCAPM_UNINIT	Application failed to call **TCAPInitMgmtAPI** prior to this call.

Details

All TCAP alarm messages are sent to the *txalarm* utility.

The following table defines the TCAP alarm levels and their recommended use:

Alarm level	Description
TCAP_ALARMLVL_DIS	All alarms are disabled. This alarm level is not recommended for use.
TCAP_ALARMLVL_DFLT	Default alarm level. Service impacting events such as application/SSN unavailable, resource failures, and application interface violations are logged.
TCAP_ALARMLVL_DEBUG	All default level alarms plus detailed message encoding or decoding diagnostics to troubleshoot individual application or transaction problems. Use this alarm level for application development.
TCAP_ALARMLVL_DETAIL	All debug level alarms plus some normal events to help isolate network or application problems. Do not use this alarm level under production load conditions.

TCAPGenCfg

Sends the TCAP general configuration parameters to the TX board.

Prototype

S16 **TCAPGenCfg** (U8 ***board***, TcapGenCfg ****cfg***)

Argument	Description
board	TX board number to which this request is directed.
cfg	Pointer to the address of a general configuration parameters buffer. The buffer's format is specified in **TCAPInitGenCfg**.

Return values

Return value	Description
TCAP_SUCCESS	
TCAPM_BADPARAM	A parameter is out of range, most likely because the general configuration parameters buffer was not initialized using **TCAPInitGenCfg**.
TCAPM_BOARD	***board*** is out of range.
TCAPM_CFGDUP	General configuration has already been sent to the board.
TCAPM_DRIVER	Error occurred accessing the driver.
TCAPM_FAILED	Task on the TX board reported a failure.
TCAPM_MAXSAPS	Value specified for maxSaps is out of range.
TCAPM_NULLPTR	Null pointer was specified for ***cfg***.
TCAPM_RANGE	Value is out of range.
TCAPM_TIMEOUT	Request timed out.
TCAPM_UNINIT	Application failed to call **TCAPInitMgmtAPI** prior to this call.

Details

This function can be called only once per download of the TX board. It must be called before any TCAP SAPs are configured.

cfg must be initialized with **TCAPInitGenCfg** prior to this call.

TCAPGenStatus

Retrieves the TCAP general status structure.

Prototype

S16 **TCAPGenStatus** (U8 ***board***, TcapGenStatus ****status***)

Argument	Description
board	TX board number.

status	Pointer to the TcapGenStatus structure:
	`/* TCAP General statistics structure */`
	`typedef struct TCAP_Gen_Stats`
	`{`
	`  U8  haSt;  /* high availability state */`
	`  U8  mcSt;  /* mate connection state  */`
	`} TcapGenStatus;`
	Refer to the Details section for more information.

Return values

Return value	Description
TCAP_SUCCESS	
TCAPM_BOARD	***board*** is out of range.
TCAPM_DRIVER	Error occurred accessing the driver.
TCAP_FAILED	Task on the TX board reported a failure.
TCAPM_NULLPTR	Null pointer was specified for ***status***.
TCAPM_TIMEOUT	Request timed out.
TCAPM_UNINIT	Application failed to call **TCAPInitMgmtAPI** prior to this call.

Details

haSt has the following valid values:

Value	Description
0	ST_HAST_STARTING
1	ST_HAST_STANDALONE
2	ST_HAST_PRIMARY
3	ST_HAST_BACKUP

mcSt has the following valid values:

Value	Description
0	ST_MCST_ISOLATED
1	ST-MCST_CONNECTED

TCAPGetGenCfg

Retrieves the current values for the general configuration parameters from the TX board.

Prototype

S16 **TCAPGetGenCfg** (U8 ***board***, TcapGenCfg ****cfg***)

Argument	Description
board	TX board number to which this request is directed.
cfg	Pointer to the address of a general configuration parameters buffer. The buffer's format is specified in **TCAPInitGenCfg**.

Return values

Return value	Description
TCAP_SUCCESS	
TCAPM_BOARD	***board*** is out of range.
TCAPM_DRIVER	Error occurred accessing the driver.
TCAPM_FAILED	Task on the TX board reported a failure.
TCAPM_NULLPTR	Null pointer was specified for ***cfg***.
TCAPM_TIMEOUT	Request timed out.
TCAPM_UNINIT	Application failed to call **TCAPInitMgmtAPI** prior to this call.

TCAPGetSapCfg

Retrieves the current configuration parameter values for a specific TCAP SAP.

Prototype

S16 **TCAPGetSapCfg** (U8 ***board***, TcapSapCfg ****cfg***, U16 ***sapId***)

Argument	Description
board	TX board number to which this request is directed.
cfg	Pointer to the address of the SAP configuration parameters buffer. The buffer's format is specified in **TCAPInitSapCfg**.
sapId	Service access point being defined.

Return values

Return value	Description
TCAP_SUCCESS	
TCAPM_BADSAP	SAP ID is either out of range or the call was made prior to calling **TCAPGenCfg**.
TCAPM_BOARD	***board*** is out of range.
TCAPM_DRIVER	Error occurred accessing the driver.
TCAPM_FAILED	Task on the TX board returned a failure.
TCAPM_NULLPTR	Null pointer was specified for ***cfg***.
TCAPM_TIMEOUT	Request timed out.
TCAPM_UNINIT	Application failed to call **TCAPInitMgmtAPI** prior to this call.

TCAPInitGenCfg

Initializes a TCAP general configuration buffer to default configuration values that can be passed to **TCAPGenCfg**.

Prototype

S16 **TCAPInitGenCfg** (TcapGenCfg ***cfg***)

Argument	Description

cfg	Pointer to the TCAP general configuration structure to be initialized:
	`typedef struct Tcap_Gen_Cfg`
	`{`
	`  S16  maxSaps;  /* Max Number of TCAP Saps */`
	`  S16  maxDlgs;  /* Max number of dialogs; system-wide */`
	`  S16  maxInvs;  /* Max number of invokes; system-wide */`
	`  S16  timeRes;  /* Timer Resolution - for internal use */`
	`  PDesc smPst;  /* not used */`
	`  U8  alarmLvl;  /* Alarm level */`
	`  U8  minTidLen;  /* Minimum transaction ID length`
	`  * (ITU only) */`
	`  U8  haState;  /* high availability state (ouput only) */`
	`  U8  mcState;  /* mate connection state (output only) */`
	`  U8  traceData;  /* enables/disables TCAP packet trace */`
	`  U8  fill;  /* spare for alignment */`
	`  U16  memThresh1; /* congestion onset level 1 mem percent */`
	`  U16  memThresh2; /* congestion onset level 2 mem percent */`
	`  U16  MemThresh3; /* congestion onset level 3 mem percent */`
	`} TcapGenCfg;`
	Refer to the Details section for more information.

Return values

Return value	Description
TCAP_SUCCESS	
TCAPM_NULLPTR	Null pointer was specified for ***cfg***.

Details

The application can change the default values within the specified range for any fields other than those denoted as internal or unused prior to calling **TCAPGenCfg** to send the configuration to the TCAP layer.

TcapGenCfg structure members not listed in the following table are either unused or for internal use only. These fields are set to correct values by **TCAPInitGenCfg** and must not be overridden by the application.

Default values for the TcapGenCfg structure that can be overridden by the calling application are listed in the following table:

Field	Range	Default value	Description
maxSaps	1 - 64	4	Maximum number of TCAP user SAPs (subsystem number or protocol variant) that can be defined.
maxDlgs	1 - 32767	256	Maximum number of outgoing and incoming TCAP transactions that can be pending at one time.
maxInvs	1 - 32767	256	Maximum number of outgoing and incoming TCAP invoke operations that can be pending at one time.
alarmLvl	TCAP_ALARMLVL_DIS TCAP_ALARMLVL_DFLT TCAP_ALARMLVL_DEBUG TCAP_ALARMLVL_DETAIL	TCAP_ALARMLVL_DFLT	Level of alarms generated by the TCAP layer. **TCAPAlarmControl** provides information on alarm levels.
minTidLen	1 - 4	1	Minimum transaction ID length (ITU only). If an out of range value is specified, the value is forced to 1 without a reported error.
haState	ST_HAST_STARTING ST_HAST_STANDALONE ST_HAST_PRIMARY ST_HAST_BACKUP	N/A	Current TCAP layer state returned by **TCAPGetGenCfg**. It is ignored by **TCAPGenCfg**.
mcState	ST_MCST_ISOLATED ST_MCST_CONNECTED	N/A	Current TCAP layer state of communication with mated TCAP when deployed as a redundant pair. Returned by **TCAPGetGenCfg**. The layer state is ignored by **TCAPGenCfg**.
traceData	TCAP_BUFTRACE_OFF TCAP_BUFTRACE_ON	TCAP_BUFTRACE_OFF	Enables and disables tracing of TCAP packets to the *ss7trace* utility.
memThresh1	1 - 99	20	Percentage of memory available to TCAP below which inbound and outbound congestion level 1 is triggered.
memThresh2	1 - 99	15	Percentage of memory available to TCAP below which inbound and outbound congestion level 2 is triggered.
memThresh3	1 - 99	10	Percentage of memory available to TCAP below which inbound and outbound congestion level 3 is triggered.

TCAPInitMgmtAPI

Initializes TCAP management and provides access to the TX board.

Prototype

S16 **TCAPInitMgmtAPI** (U8 ***board***, U8 ***srcEnt***, U8 ***srcInst***)

Argument	Description
board	TX board number to which this request is directed.
srcEnt	Calling application entity ID.
srcInst	Calling application instance ID.

Return values

Return value	Description
TCAP_SUCCESS	
TCAPM_BOARD	***board*** is out of range.
TCAPM_DRIVER	Error occurred accessing the driver.
TCAPM_TOOMANY	Too many applications initialized the board concurrently.

Details

This function must be called before any other TCAP management functions are called.

If the same application uses both the TCAP service functions and the TCAP management functions, specify separate entity IDs when the two APIs are initialized.

TCAPInitSapCfg

Builds a default TCAP user SAP configuration buffer that can be passed to **TCAPSapCfg**.

Prototype

S16 **TCAPInitSapCfg** (U8 ***board***, S16 ***spId***, S16 ***swProt***, TcapSapCfg ****cfg***)

Argument	Description
board	TX board number to which this request is directed. This parameter also identifies the TCAP layer instance ID.
spId	Index number of the TCAP SAP being defined. Valid range is 0 to (TcapGenCfg.maxSaps - 1).
swProt	Protocol selector switch. Refer to the Details section for more information.

cfg	Pointer to the address of the TCAP SAP configuration parameters buffer:
	`typedef struct Tcap_Sap_Cfg`
	`{`
	`  S16 swtch; /* Protocol selector switch */`
	`  U8 selectorUser; /* selector for TCAP User */`
	`  U8 spare1; /* spare for alignment */`
	`  MemoryId memUser; /* Memory ID - not used */`
	`  U8 priorUser; /* priority - not used */`
	`  U8 routeUser; /* route - not used */`
	`  TmrCfg t1; /* default invocation timer */`
	`  TmrCfg t2; /* wait for rejection timer */`
	`  U8 seqTimer; /* duration to maintain SLS for`
	`  * sequential delivery */`
	`  U8 selectorSP; /* selector for SCCP */`
	`  MemoryId memSP; /* memory ID/SCCP - not used */`
	`  U16 procIdSP; /* processor Id -not used */`
	`  U8 entSP; /* SCCP entity ID = ENT_SCCP */`
	`  U8 instSP; /* SCCP inst. ID = board num. */`
	`  U8 priorSP; /* SCCP priority - not used */`
	`  U8 routeSP; /* SCCP route - not used */`
	`  S16 spIdSP; /* SCCP SAP ID for this TCAP SAP */`
	`  U8 altParamLen; /* Alternate Parameter Length`
	`  * Calculation */`
	`  U8 chkpt; /* default checkpoint strategy for`
	`  /* this application */`
	`  U8 addrOverride; /* application overrides SCCP`
	`  /* address */`
	`  U8 allowInvkEnd; /* allow invoke component in end`
	`  /* message */`
	`  S16 qThresh1; /* queue size triggering congestion`
	`  /* level 1 */`
	`  S16 qThresh2; /* queue size triggering congestion`
	`  /* level 2 */`
	`  S16 qThresh3; /* queue size triggering congestion`
	`  /* level 3 */`
	`} TcapSapCfg;`
	Refer to the Details section for more information.

Return values

Return value	Description
TCAP_SUCCESS	
TCAPM_BOARD	***board*** is out of range.
TCAPM_NULLPTR	Null pointer was specified for ***cfg***.
TCAPM_SWTYPE	Protocol selector switch type is invalid.

Details

swProt identifies one of the following TCAP protocol variants used on the specified SAP:

Value	Description
1	TCAP_SW_ITU88
2	TCAP_SW_ITU92
3	TCAP_SW_ANS88
4	TCAP_SW_ANS92
5	TCAP_SW_ITU97
6	TCAP_SW_ANS96

The application can change the default values within the specified range for any fields other than those denoted as internal or unused prior to calling **TCAPSapCfg** to send the configuration block to the TCAP layer.

TcapSapCfg structure members not listed in the following table are either unused or for internal use only. These fields are set to correct values by **TCAPInitSapCfg** and must not be overridden by the application.

Default values for the TcapSapCfg structure that can be overridden by the calling application are listed in the following table:

Field	Range	Default value	Description
swtch	1 - 6	***swProt*** parameter	TCAP protocol variant to be used on this SAP.
t1	1 - 65535	60	Default invocation timer, in seconds.
t2	1 - 65535	60	Time to wait for reject of non-invoke component, where applicable, in seconds.
seqTimer	1 - 255	60	Duration to request SCCP to maintain signaling link selector (SLS) when sequential delivery is required.
instSP	1 - 8	***board*** parameter	SCCP task instance ID, always equal to the board number.
spIdSP	0 - 32766	***spId*** parameter	SCCP SAP ID (from the SCCP configuration) to map this TCAP SAP onto. By default, set to the same value as the TCAP SAP ID.
altParamLen	0 or 1	0	Used for ANSI TCAP protocols only. 0 = Uses the normal method of deriving a component parameter length from the component length field. 1 = Uses an alternate method of obtaining the parameter length from the set or sequence tag and length.
chkpt	CHKPT_NONE CHKPT_SEND CHKPT_ALL	CHKPT_NONE	Checkpointing option for this SAP when deployed in a redundant configuration. Refer to *Transaction checkpointing* on page 25.
addrOverride	0 or 1	0	If set to 1, the application provides a called party address in a transaction response message. By default, TCAP uses a calling party address from the incoming message that initiated the transaction.
allowInvkEnd	0 or 1	0	If set to 1, the application includes an invoke component in the end transaction message.
qThresh1	1 - 32766	600	Number of inbound messages queued to the application before congestion level 1 is triggered.
qThresh2	1 - 32766	900	Number of inbound messages queued to the application before congestion level 2 is triggered.
qThresh3	1 - 32766	1200	Number of inbound messages queued to the application before congestion level 3 is triggered.

TCAPSapCfg

Sends a TCAP service access point (SAP) configuration parameter block to the specified TX board to define or update the configuration for a specific TCAP SAP.

Prototype

S16 **TCAPSapCfg** (U8 ***board***, TcapSapCfg ****cfg***, U16 ***sapId***)

Argument	Description
board	TX board number to which this request is directed.
cfg	Pointer to the address of the SAP configuration parameters buffer. The buffer's format is specified in **TCAPInitSapCfg**.
sapId	Service access point being defined.

Return values

Return value	Description
TCAP_SUCCESS	
TCAPM_BADSAP	SAP ID is either out of range or the call was made prior to calling **TCAPGenCfg**.
TCAPM_BOARD	***board*** is out of range.
TCAPM_DRIVER	Error occurred accessing the driver.
TCAPM_FAILED	Task on the TX board returned a failure.
TCAPM_NULLPTR	Null pointer was specified for ***cfg***.
TCAPM_SWTYPE	Protocol selector switch type is invalid.
TCAPM_TIMEOUT	Request timed out.
TCAPM_UNINIT	Application failed to call **TCAPInitMgmtAPI** prior to this call.

Details

This function can be called any time after the general configuration is downloaded to the TX board using **TCAPGenCfg** but before any application attempts to bind to this SAP for transaction processing.

An existing TCAP SAP can be redefined to change one or more of its configuration parameters by calling **TCAPSapCfg** a second time, but only if there is no application currently bound to the SAP.

cfg must be initialized with **TCAPInitSapCfg** prior to this call.

TCAPSapStats

Retrieves and optionally resets the statistics for a specified TCAP service access point (SAP).

Prototype

S16 **TCAPSapStats** (U8 ***board***, S16 ***sapID***, TcapSapStats ****stats***, U8 ***reset***)

Argument	Description
board	TX board number to which this request is directed.
sapId	Index number of the target TCAP SAP.
stats	Pointer to the address of a caller's TCAP SAP statistics buffer where statistics are to be returned. Refer to the Details section for more information.
reset	If non-zero, statistics are reset to zero after retrieving.

Return values

Return value	Description
TCAP_SUCCESS	
TCAPM_BADSAP	***sapId*** is either out of range or the call was made prior to calling **TCAPGenCfg**.
TCAPM_BOARD	***board*** is out of range.
TCAPM_DRIVER	Error occurred accessing the driver.
TCAP_FAILED	Task on the TX board reported a failure.
TCAPM_NULLPTR	Null pointer was specified for ***stats***.
TCAPM_TIMEOUT	Request timed out.
TCAPM_UNINIT	Application failed to call **TCAPInitMgmtAPI** prior to this call.

Details

The TcapSapStats structure is shown here. Counts of type S32 represent the number of event occurrences since last cleared. Counts of type TcapEvCnt include both a count of events plus a time stamp of the first occurrence of the event, displayed in hundredths of seconds since the last boot.

```
typedef struct Tcap_Ev_Cnt
{
  S32        cnt;        /* event count                      */
  U32        first;      /* when it first happened           */
} TcapEvCnt;

typedef struct Tcap_Sap_Stats
{
  S16       swtch;        /* protocol variant switch          */
  S16       fill;         /* fill for alignment               */
  S32       openTrans;    /* number of open transactions      */
  S32       frmTx;        /* frames transmitted               */
  S32       invTx;        /* invoke components transmitted    */
  S32       resTx;        /* result components transmitted    */
  S32       rejTx;        /* reject components transmitted    */
  S32       errTx;        /* error components transmitted     */
  S32       uniTx;        /* unidirectional msgs xmitted      */
```

```
  S32       beginTx;      /* begin messages transmitted        */
  S32       contTx;       /* continue messages transmitted     */
  S32       endTx;        /* end messages transmitted          */
  S32       abortTx;      /* abort messages transmitted        */
  S32       qryPrmTx;     /* query with permission xmitted     */
  S32       qryNoPrmTx;   /* query w/out permission xmitted    */
  S32       conPrmTx;     /* conversation with permission
                           * xmitted                           */
  S32       conNoPrmTx;   /* conversation without permission
                           * transmitted                       */
  S32       respTx;       /* response transmitted              */
  S32       frmRx;        /* frames received                   */
  S32       invRx;        /* invoke components received        */
  S32       resRx;        /* result components received        */
  S32       rejRx;        /* reject components received        */
  S32       errRx;        /* error components received         */
  S32       uniRx;        /* unidirectional msgs received      */
  S32       beginRx;      /* begin messages received           */
  S32       contRx;       /* continue messages received        */
  S32       endRx;        /* end messages received             */
  S32       abortRx;      /* abort messages received           */
  S32       qryPrmRx;     /* query with permission rec'd       */
  S32       qryNoPrmRx;   /* query without permission rec'd    */
  S32       conPrmRx;     /* conversation with permission
                           * rec'd                             */
  S32       conNoPrmRx;   /* conversation w/out permission
                           * rec'd                             */
  S32       respRx;       /* response received                 */
  S32       drop;         /* frames dropped                    */
  TcapEvCnt urPkg;        /* unrecognized package type         */
  TcapEvCnt inTrn;        /* incorrect transaction portion     */
  TcapEvCnt bdTrn;        /* badly structured transaction
                           * portion                           */
  TcapEvCnt urTrn;        /* unrecognized transaction ID       */
  TcapEvCnt prTrn;        /* permission to release problem     */
  TcapEvCnt ruTrn;        /* resource unavailable              */
  TcapEvCnt urCmp;        /* general - unrecognized comp.      */
  TcapEvCnt inCmp;        /* general-incorrect component
                           * portion                           */
  TcapEvCnt bdCmp;        /* general - badly structured
                           * component portion                 */
  TcapEvCnt dupId;        /* invoke - duplicate Invoke Id      */
  TcapEvCnt urOp;         /* invoke - unrecognized op code     */
  TcapEvCnt inPrm;        /* invoke - incorrect parameters     */
  TcapEvCnt iurId;        /* invoke - unrecognized correlation
                           * ID                                */
  TcapEvCnt rurId;        /* ret. result - unrecognized correl.
                           * ID                                */
  TcapEvCnt uxRes;        /* ret. result - unexpected ret.
                           * result                            */
  TcapEvCnt eurId;        /* ret. error - unrecognized correl.
                           * ID                                */
  TcapEvCnt uxRer;        /* ret. error - unexpected return
                           * error                             */
  TcapEvCnt urErr;        /* ret. error - unrecognized error
                           * code                              */
  TcapEvCnt uxErr;        /* ret. error - unexpected error     */
  TcapEvCnt enPrm;        /* ret. error - incorrect parameter  */
  S32       outCongAbort; /* outbound transactions refused -
                           /* congestion                        */
  S32       outCongDisc;  /* outbound msgs discarded -
                           /* congestion                        */
  S32       inbCongAbort; /* inbound transactions refused -
                           /* congestion                        */
  S32       inbCongDisc;  /* inbound msgs discarded -
                           /* congestion                        */
  U8      currInbCongLvl; /* current inbound congestion level  */
  U8     currOutbCongLvl; /* current outbound congestion level */
} TcapSapStats;
```

TCAPTermMgmtAPI

Terminates the connection between the application and the TX board, releasing any resources (file descriptors) associated with TCAP management functions.

Prototype

S16 **TCAPTermMgmtAPI** (U8 ***board***)

Argument	Description
board	TX board number to which this request is directed.

Return values

Return value	Description
TCAP_SUCCESS	
TCAPM_BOARD	***board*** is out of range.
TCAPM_UNINIT	Application failed to call **TCAPInitMgmtAPI** prior to this call.

Details

Call this function once for each board the application initialized.

TCAPTraceControl

Sends a request to enable or disable tracing of TCAP protocol messages.

Prototype

S16 **TCAPTraceControl** (U8 ***board***, U32 ***flags***)

Argument	Description
board	TX board to which this request is directed.
flags	Bit map of trace facilities to turn on or off.

Return values

Return value	Description
TCAP_SUCCESS	
TCAPM_BOARD	***board*** is out of range.
TCAPM_DRIVER	Error occurred accessing the driver.
TCAPM_FAILED	Task on the TX board reported a failure.
TCAPM_TIMEOUT	Request timed out.
TCAPM_UNINIT	Application failed to call **TCAPInitMgmtAPI** prior to this call.

Details

TCAPTraceControl is currently not implemented.

TCAP supports only protocol buffer tracing (dumps of all TCAP protocol messages sent or received).

0x00	TCAP_BUFTRACE_OFF
0x01	TCAP_BUFTRACE_ON

7 Demonstration programs and utilities

Summary of the demonstration programs and utilities

NMS TCAP provides the following demonstration programs and utilities:

Program	Description
find800	Implements a single request and response transaction using the TCAP layer.
tcapcfg	Downloads the TCAP configuration to the TX board at boot time.
tcapmgr	Monitors and manages the status of the TCAP layer.

Request and response transaction: find800

Implements a single request and response transaction using the TCAP layer.

Separate samples are included for ITU-T TCAP and ANSI TCAP.

This program...	Uses...	Defaults to...	Optionally supports...
ansi800	ANSI TCAP messages	24-bit addressing	14-bit ITU addressing
itu800	ITU-T TCAP messages	14-bit ITU-T addressing	24-bit ANSI addressing

find800 uses NMS TCAP to send and receive 800 number translation requests. *find800* can act as an 800 number server, or as an 800 number client requesting an 800 number translation. Both utilities can be found in the *tektx**samples**tcap*\ directory.

Usage

```
find800 [options] pointcode:subsystem phonenum
```

Requirements

- A computer with a TX board installed
- Windows or UNIX
- Natural Access
- NMS SS7

Procedure

To run *find800*:

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>From the command line prompt, navigate to the tektx\samples\tcap\find800 directory under Windows or the /usr/bin directory under UNIX.</td></tr>
<tr><td>2</td><td>Enter the following command:
<code>find800 [options] pointcode:subsystem phonenum</code>
where options include:
<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>-b board</td><td>TX board number.
Default = 1. Valid range is 1 - 8.</td></tr>
<tr><td>-p sapno</td><td>Service access point ID.
Default = 0. Valid range is 0 - 255.</td></tr>
<tr><td>-n number</td><td>Subsystem number.
Default = 254. Valid range is 0 - 255.</td></tr>
<tr><td>-i iterations</td><td>Number of times a transaction is repeated.
Default = 1. Valid range is 0 - 32000.</td></tr>
<tr><td>-j delay</td><td>Delay, in ms, between transaction repetition.
Default = 5000. Valid range is 1 - 65536.</td></tr>
<tr><td>-d</td><td>Activates detailed dump of each sent or received packet.</td></tr>
<tr><td>-t</td><td>Uses ITU addressing. The ansi800 program defaults to ANSI addressing.</td></tr>
<tr><td>-a</td><td>Uses ANSI addressing. The itu800 program defaults to ITU addressing.</td></tr>
<tr><td>-s</td><td>Causes find800 to act as an 800 number server. The find800 program acts as a client by default.</td></tr>
</table>
pointcode:subsystem specifies the pointcode and subsystem number of the 800 number server that is used by clients.
phonenum indicates the 800 number to be translated that is used by clients.</td></tr>
</table>

Note: If multiple instances of *find800* are bound to the same TX board, the SAP ID (-s) and the subsystem number (-n) must be unique for each instance.

800 number server

To start *find800* as an 800 number server:

Step	Action

1	Enter the following command:
	`find800 -b 1 -p 0 -n 255 -s`
	In this case, *find800* binds to TX board 1, uses SAP ID 0, and uses subsystem number 255. Since -s is specified, *find800* acts as a server and waits for an 800 number request to arrive.
	When a request arrives, *find800* takes the received 800 number and compares it to the numbers found in the *numbers.800* file.
	Note: The *numbers.800* file must be in the same directory as the *find800.exe* file.
	The *numbers.800* file looks like this:
	`[800 Numbers]`
	`8001234567=3122456789`
	`8004561234=8477069700`
	Additional 800 numbers can be added, as long as they are listed after the [800 Numbers] section header and they conform to the following syntax:
	`800nnnnnnn=yyyyyyyyyy`
	If a matching 800 number is found, the *FIND800* server returns the translated number in a RETURN_RESULT [***last***] component.
	If no matching 800 number is found, the *find800* server returns a RETURN_ERROR component.
	The *find800* server continues to listen for and respond to requests indefinitely.

2	To stop the server, press any key.

800 number client

To start *find800* as a client:

Step	Action
1	Enter the following command: `find800 -b 2 -p 1 -n 254 1.1.1:255 8001234567` In this case, *find800* binds to TX board 2, uses SAP ID 1, and uses subsystem number 254. Since -s is not specified, *find800* acts as a client, and immediately sends an 800 number request to pointcode 1.1.1, subsystem 255. The 800 number to be translated is 8001234567. After sending the 800 number request, *find800* waits for a response. After a response is received, *find800* continues to run, but no further requests are sent.
2	To stop the client, press any key.

Troubleshooting

If TCAP messages are not sent, check the following:

- By default, the TCAP configuration is set up for ANSI-protocol messages. Ensure that you are using the *ansi800* version of *find800*.
- Structures used by TCAP must be packed on one-byte boundaries. The default in most compilers is packing on 8-byte boundaries. For both Windows and UNIX systems, the -Zp compiler flag must be set. The sample code makefile shows how this flag is properly set.
- By default, the *find800* demonstration program uses subsystem number 254. However, the SCCP layer is initially configured with only subsystem numbers 3 and 4. Use the -n 3 command line option on both the client and the server of *find800*.

  The following code is an example of starting *find800* as a client using subsystem number 3:

```
find800 -b 2 -p 1 -n 3 1.1.1:3 8001234567
```

Using the TCAP ITU-T protocol

To use the TCAP ITU protocol, modify the *tcapcp1.cfg* file. Change the SWITCH_TYPE parameter to ITU88, ITU92, or ITU97 and modify as many SAP IDs as needed:

```
#
# User SAP configuration for 1st application
#
USER_SAP        0       # Sap number start at 0
SWITCH_TYPE     ITU92   # one of ITU92, ITU88, ANSI92, ANSI88
END                     # User application 0
```

Note: ANSI-style point code addressing is still used (1.1.1).

If two TX boards are used, modify the *tcapcp2.cfg* file.

Adding subsystem numbers

To define new subsystem numbers, modify the *sccpcp1.cfg* file. The following code is an example of a subsystem definition section:

```
#define all subsystems of interest at 1.1.1 (up to 8)
 SSN        3          # first subsystem at 1.1.2
 SSN_SNR    TRUE       # normal routed
 SSN_ACC    TRUE       # initially accessible
#SSN_BPC    x.y.z      # this subsystem not currently replicated
        # concerned point codes - other nodes to be notified when
        # status of this SSN at this node changes - must have a
        # route for any point code listed here
#CONC_PC    q.r.s      # 1st concerned point code
#CONC_PC    q.r.t      # 2nd concerned point code
END                     # of route 1.1.2, SSN 3
```

Either change the SSN field or copy the definition section and define new subsystem numbers. If two TX boards are used, modify the *sccpcp2.cfg* file.

TCAP configuration utility: tcapcfg

Downloads the TCAP configuration to the TX board at boot time.

Once the TCAP configuration file is created, run the TCAP configuration (*tcapcfg*) utility to download it to scan the ASCII text file and download the configuration to the TCAP task on the TX board.

Usage

```
tcapcfg options
```

Requirements

- A computer with a TX board installed
- Windows or UNIX
- Natural Access
- NMS SS7

Procedure

To run *tcapcfg*:

Step	Action
1	From the command line prompt, navigate to the *\tektx\samples\tcap\tcapcfg* directory under Windows or the */usr/bin* directory under UNIX.
2	Enter the following command: `tcapcfg options` where ***options*** include: **Options** / **Description** -b ***board*** — Board number to which the TCAP configuration is downloaded. Default = 1. -f ***filename*** — Name and location of the TCAP configuration file to be downloaded. The TCAP configuration program scans the information in the ASCII file (specified with the -f option) and downloads this information to the task on the TX board.

Details

The TCAP configuration utility is available in both source code and executable formats. Use *tcapcfg* if you want your application to load the TCAP configuration to the TX board.

TCAP layer status: tcapmgr

After downloading the TCAP configuration to the TX board with *tcapcfg*, run the TCAP manager (*tcapmgr*) program to monitor the status of the TCAP layer. The TCAP manager provides a command line interface that enables an application to set alarm levels, trace buffers, and view and reset TCAP statistics.

Usage

```
tcapmgr -b board
```

Requirements

- A computer with a TX board installed
- Windows or UNIX
- Natural Access
- NMS SS7

Procedure

To run *tcapmgr*:

Step	Action
1	From the command line prompt, navigate to the *\tektx\samples\tcap\tcapmgr* directory under Windows or the */usr/bin* directory under UNIX.
2	Enter the following command: `tcapmgr -b board` where ***board*** is the TX board number where the TCAP layer is loaded. The *tcapmgr* program supports the following commands:

Command	Description
ALARMLVL ***options***	Sets the current alarm output level. Valid range for ***options*** is 0 - 3.
CONFIG SAP ***sapno*** | GEN	Displays current general or SAP configuration parameter values.
STATS ***sapno [RESET]***	Retrieves statistics on an application service access point (***sapno***) and optionally resets them to zero (***RESET***) after fetch.
STATUS	Retrieves the general status of the TCAP task.
TRACE ***ON | OFF***	Turns buffer tracing on or off. This feature is not currently supported.
BOARD ***board***	Switches to a new target board (***board***).
? ***[COMMAND]***	Lists available commands or parameters of a specific command (***COMMAND***).
Q	Quits the application.

Details

The TCAP manager program is available in both source code and executable formats. The source code demonstrates the use of TCAP management for developers who want to integrate management of the SS7 TCAP layer into their own configuration management systems.

8 Parameter and event structure overview

Data types

NMS TCAP uses the following conventions for data types:

TCAP type	C implementation	Description
U8	unsigned char	unsigned 8-bit quantity
U16	unsigned short	unsigned 16-bit quantity
S16	short	signed 16-bit quantity
U32	unsigned long	unsigned 32-bit quantity
S32	long	signed 32-bit quantity

Point codes

Signaling point codes are represented as 32 bit unsigned integers (type U32) of which either the least significant 14 bits or 24 bits (depending on the specific configuration) are used. For example, an ANSI point code represented by the (decimal) string 1.4.7 is encoded as (hex) 0x00010407.

TCAP octet strings

TCAP octet strings represent variable length octet strings such as ITU-T application context names, operation codes, and problem codes:

```
typedef struct Tcap_Octet_Str
{
  U16 length;
  U8  fill;                    /* fill for alignment */
  U8  fill2;                   /* fill for alignment */
  U8  octStr[TCAP_MAX_OCTSTR];
} TcapOctetStr;
```

For optional octet strings, the length field is set to zero to indicate that the parameter is not included. Currently, the maximum length of octet strings using this structure is set to 64 bytes.

TCAP component IDs

TCAP component IDs are used in the component definitions to represent invoke IDs, linked IDs (ITU-T), and correlation IDs (ANSI).

They consist of a flag to indicate if the ID is present or not (for example, for optional IDs). If the ID is present, an ID value is also specified.

For the present flag, use the present or not present identifiers defined in *sccpapi.h*.

```
typedef struct Tcap_CompId
{
  U8  present;     /* component ID present/not present */
  U8  compId;      /* component ID value               */
  U8  fill;        /* fill for alignment               */
  U8  fill2;       /* fill for alignment               */
} TcapCompId;
```

Global titles

Strings of digits, such as a telephone number or a mobile identification number, that routes TCAP transactions when the actual destination point code is unknown or can change. A global title is translated into a destination point code/SSN by the SCCP layer. Translations are configured in the SCCP configuration database. For more information, refer to *Global title translation* on page 33.

The global title must be BCD-encoded. The global title 8471234567 is BCD encoded as:

```
tInfo.cdAddr.glTitle[0] = 0x48;
tInfo.cdAddr.glTitle[1] = 0x17;
tInfo.cdAddr.glTitle[2] = 0x32;
tInfo.cdAddr.glTitle[3] = 0x54;
tInfo.cdAddr.glTitle[4] = 0x76;
```

The original global title was 10 digits long. BCD-encoded, it is five bytes in length. This is used as the global title length.

```
tInfo.cdAddr.glTitleLen = 5;
```

Global titles are encoded as follows:

Octet 1	2nd address digit	1st (most significant) address digit
...	...	...
Octet n	m + 1th address digit or filler	mth address digit

Each digit is encoded with the following bit pattern:

Bit pattern	Digit/signal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	spare
1011	code 11
1100	code 12
1101	spare
1110	spare
1111	ST

9 Common data structures

Common data structures summary

This section describes the following common data structures:

- SCCP address structure
- SCCP quality of service (QOS) structure
- TCAP transaction information structure
- TCAP dialog section structure

SCCP address structure

The SCCP address structure represents a called party (destination) address and a calling party (originating) address. The following declarations are found in the *sccpapi.h* file:

```
typedef struct Tcap_Sp_Addr /* SCCP Address                */
{
  U8  presind;             /* presence indicator           */
  U8  spare1;              /* spare for alignment          */
  S16 swType;              /* switch type (ANSI/ITU-T)     */
  U8  subsystemInd;        /* subsystem indicator          */
  U8  pointCodeInd;        /* point code indicator         */
  U8  glTitleInd;          /* global title indicator       */
  U8  routingInd;          /* routing indicator            */
  U8  natIntInd;           /* national/international ind.  */
  U8  subsystem;           /* subsystem number             */
  U32 pointCode;           /* point code                   */
  U8  glTransType;         /* global title translation type */
  U8  encoding;            /* address encoding scheme      */
  U8  numPlan;             /* numbering plan               */
  U8  natAddrInd;          /* nature of address indicator  */
  U8  spare2;              /* spare for alignment          */
  U8  glTitleLen;          /* length of global title       */
  U8  glTitle[MAX_GLT_SZ]; /* Global Title                 */
} SccpAddr;
```

The fields in the sccpAddr structure are encoded as follows:

Field	Value
presind	0 = NOT_PRESENT Field not present in message 1 = PRESENT Field present in message
swtype	1 = SW_ITU ITU-T address 2 = SW_ANSI ANSI address
subsystemInd	0x00 = SUBSYS_NONE No subsystem number in address 0x01 = SUBSYS_INC Address contains subsystem number
pointCodeInd	0x00 = PTCODE_NONE No point code in address 0x01 = PTCODE_INC Address contains point code
glTitleInd	0x00 = GLT_NONE No global title in address 0x01 = GLT_TT_NP_E ANSI global title includes translation type, numbering plan and encoding 0x02 = GLT_TT ANSI global title includes translation type only 0x01 = GLT_ITU_FMT1 ITU global title includes encoding and nature of address 0x02 = GLT_ITU_FMT2 ITU global title includes translation type only 0x03 = GLT_ITU_FMT3 ITU global title includes translation type, numbering plan, and encoding 0x04 = GLT_ITU_FMT4 ITU global title includes translation type, numbering plan, encoding, and nature of address
routingInd	0x00 = ROUTE_GLT Route by global title only 0x01 = ROUTE_PC_SN Route by point code and subsystem number
natIntInd	0x00 = ADDRIND_INT International address indicator 0x01 = ADDRIND_NAT National address indicator

Field	Value
subsystem	0x00 = SUBSYS_NONE Subsystem unknown or not used 0x01 = SUBSYS_SCCPMGMT SCCP management subsystem 0x03 = SUBSYS_ISUP ISUP subsystem 0x04 = SUBSYS_OMAP Operations, maintenance, and administration 0x05 = SUBSYS_MAP Mobile application part 0x06 = SUBSYS_HLR Home location register 0x07 = SUBSYS_VLR Visitor location register 0x08 = SUBSYS_MSC Mobile switching center 0x09 = SUBSYS_EIR Equipment identification register 0x0A = SUBSYS_AUTH Authentication center Other values in range 0x0B - 0xFF are also allowed.
pointCode	A 32 bit quantity of which the least significant 24 bits (ANSI) or the least significant 14 bits (ITU-T) are used. For example, an ANSI point code represented by the decimal string 1.4.7 is encoded as hex 0x00010407. If a point code is not included in the called address, the default point code is used.
glTransType	Translation type when the global title indicator field (glTitleInd) specifies that the global title includes translation type. Any 8-bit value [0x00 - 0xFF] is allowed.
encoding	Specifies whether the number of digits in the addrSig field is even or odd. If the number of digits is even, the last octet contains two digits. If the number of digits is odd, the last octet contains only one digit and the most significant four bits are not used. 0x00 = ENC_UNKNOWN Encoding unknown 0x01 = ENC_BCD_ODD BCD, odd number of digits 0x02 = ENC_BCD_EVEN BCD, even number of digits
numPlan	0x00 = NP_UNK Unknown 0x01 = NP_ISDN ISDN/telephony - E.164/E.163 0x02 = NP_TEL Telephony numbering - E.163 0x04 = NP_TELEX Telex numbering - Recommendation F.69 0x05 = NP_MARITIME Maritime mobile numbering 0x06 = NP_LANDMOB Land mobile numbering 0x08 = NP_NATIONAL National standard numbering 0x09 = NP_PRIVATE Private numbering 0x0f = NP_EXT Reserved for extension
	0x01 = NATIND_SUBS Subscriber number 0x03 = NATIND_NATL National number
glTitleLen	Contains the byte length of the encoded global title. For example, a 10-digit global title, after BCD encoding, would be five bytes long, so glTitleLen would be set to five bytes.
	Encoded global title.

SCCP quality of service (QOS) structure

Specifies a requested quality of service from the SCCP layer:

```
typedef struct Tcap_Sccp_Qos
{
  U8  priority;    /* message priority (lowest) 0..3 (highest)         */
  U8  retOpt;      /* action to take if msg undeliverable              */
  U8  seqDlvy;     /* sequential deliver required?                     */
  U8  spare;       /* spare for alignment                              */
} TcapSccpQos;
```

The fields in the TcapSccpQos structure are encoded as follows:

Field	Value
priority	Message priority range 0 (lowest) to 3 (highest).
retOpt	0x08 = MSG_RETURN Return message on error
seqDlvy	1 = TCAP_QOS_SEQDEL Sequential delivery required, use SCCP class 1

TCAP transaction information structure

Holds the transaction-level information regarding an outgoing transaction request or an incoming transaction message:

```
typedef struct Tcap_Trans_Info
{
  U8          msgType;     /* transaction message/package type          */
  U8          abortCause;  /* cause of an abort by TCAP layer, valid
                            * only if msgType is TCAP_ANSI_PABORT or
                            * TCAP_P_ABORT                              */
  U8          preArgEnd;   /* pre-arranged end? 0=no, 1=yes             */
  U8          chkpt;       /* Checkpoint behavior                       */
  U16         inactvTimer; /* Inactivity timer                          */
  U16         spare1;      /* spare for alignment                       */
  U32         suDlgId;     /* service user dialog ID                    */
  U32         spDlgId;     /* service provider (TCAP) dialog ID         */
  TcapSpAddr  cdAddr;      /* called party address                      */
  TcapSpAddr  cgAddr;      /* calling party address                     */
  TcapSccpQos qos;         /* SCCP quality of service requested         */
} TcapTransInfo;
```

The fields in the TcapTransInfo structure are encoded as follows:

	Value
msgType	The message type identifies the TCAP transaction message that is sent or received: 1 = TCAP_BEGIN ITU-T begin message 2 = TCAP_CONTINUE ITU-T continue message 3 = TCAP_END ITU-T end message 4 = TCAP_U_ABORT ITU-T user abort message 5 = TCAP_UNI ITU-T unidirectional 6 = TCAP_QRY_PRM ANSI query with permission message 7 = TCAP_QRY_NO_PRM ANSI query without permission message 8 = TCAP_RESPONSE ANSI response message 9 = TCAP_CNV_PRM ANSI conversation with permission message 10 = TCAP_CNV_NO_PRM ANSI conversation without permission message 11 = TCAP_ANSI_UABORT TCAP ANSI user abort message 12 = TCAP_ANSI_PABORT TCAP ANSI protocol abort message 13 = TCAP_P_ABORT CCITT protocol abort message 14 = TCAP_ANSI_UNI ANSI unidirectional 15 = TCAP_LOC_IND Locally-generated comp. indication 16 = TCAP_XACTION_TIMEOUT Locally generated transaction time-out indication
abortCause	Identifies the cause of an abort initiated by the TCAP layer. Its value is valid only on incoming messages when the msgType field is TCAP_ANSI_PABORT or TCAP_P_ABORT. **P-Abort ITU-T causes** 0x00 = TCAP_ABORT_UNREC_MSG Unrecognized message 0x01 = TCAP_ABORT_UNREC_TRS Unrecognized transaction ID 0x02 = TCAP_ABORT_BAD_FRMT Malformed trans. part 0x03 = TCAP_ABORT_INC_TRANS Incorrect trans. part 0x04 = TCAP_ABORT_RESOURCE Insufficient resources 0x05 = TCAP_ABORT_ABNML_DLG Incorrect dialog portion 0x06 = TCAP_ABORT_NO_CMN_DLG Unsupported protocol version dialog portion **P-Abort ANSI causes** 0x01 = TCAP_ANSI_ABORT_UP Unrecognized package type 0x02 = TCAP_ANSI_ABORT_IN Incorrect transaction portion 0x03 = TCAP_ANSI_ABORT_BD Badly structured transaction portion 0x04 = TCAP_ANSI_ABORT_UT Unrecognized transaction ID 0x05 = TCAP_ANSI_ABORT_PR Permission to release problem 0x06 = TCAP_ANSI_ABORT_RN Resource not available

Field	Value
chkpt	If redundant TCAP is in use, this field determines the checkpoint behavior of a transaction. The default checkpoint behavior is defined in the TCAP configuration file. 0 = TCAP_NO_CHKPT Do not checkpoint this transaction 1 = TCAP_CHKPT Checkpoint this transaction 2 = TCAP_CHKPT_DEFAULT Use the default checkpoint value
inactv Timer	Determines the setting (in seconds) for the inactivity timer for the transaction. If set to 0 (zero), the transaction uses the default inactivity timer.
suDlgId	Service user dialog ID assigned by the application for this transaction. This value must be unique among all active transactions belonging to this TCAP SAP.
spDlgId	Provider dialog ID assigned by the TCAP layer for this transaction. On transactions initiated by the application, this value must be set to zero on any outgoing request sent before the first incoming transaction message is received. The first incoming message for any transaction (initiated by either end) contains the spDlgId assigned by the TCAP layer. The application should save this value to be used with subsequent requests belonging to the same transaction.
cdAddr	Destination address of a transaction message.
cgAddr	Originating address of a transaction message.
qos	Quality of service requested from the SCCP.
preArgEnd	Set to 1 if the transaction end is pre-arranged. When a transaction end is pre-arranged, there is no explicit transaction end message exchanged between the two signaling points. The transaction is cleared locally by the TCAP layer and the far signaling point is expected to do the same. If this field is set to 0 (zero), a normal transaction end is performed.

TCAP dialog section structure

Holds the optional dialog information that can be contained in a TCAP message. The dialog section is used only for ITU-T-92 or later and ANSI-96 or later protocols.

```
typedef struct Tcap_Dlg_Sect
{
  U8           dlgType;    /* type of message this dialog is for       */
  U8           resPres;    /* result field present?                    */
  U8           result;     /* dialog result: accepted/rejected perm    */
  U8           resSrc;     /* result source: user or provider (TCAP)   */
  U8           resDiag;    /* result diagnostic                        */
  U8           abrtSrc;    /* source of abort                          */
  U8           fill;       /* fill for alignment                       */
  U8           fill2;      /* fill for alignment                       */
  TcapOctetStr apConName;  /* application context name                 */
  TcapOctetStr secConInfo; /* security context name                    */
  TcapOctetStr confAlgId;  /* confidentiality algorithm ID             */
  TcapOctetStr confValue;  /* confidentiality value                    */
  U8       apConNameType;  /* security context information type        */
  U8       secConInfoType; /* confidentiality algorithm ID type        */
  U8       confAlgIdType;  /* confidentiality value type               */
  U8       confValueId;    /* confidentiality value identifier         */
} TcapDlgSect;
```

The fields in the TcapDlgSect structure are encoded as follows:

Field	Value
dlgType (ITU9x)	The dialog type identifies the dialog application PDU type. The value TCAP_DLGP_NONE indicates that the dialog portion was not present in an incoming message or should not be included in an outgoing message. 0 = TCAP_DLGP_UNK Unknown dialog portion type 1 = TCAP_DLGP_UNI Unidir. dialog portion type 2 = TCAP_DLGP_REQ Request dialog portion type 3 = TCAP_DLGP_RSP Response dialog portion type 4 = TCAP_DLGP_ABT Abort dialog portion type 1 = TCAP_DLGP_ANSI ANSI96 dialog portion type 0xFF = TCAP_DLGP_NONE Dialog portion not present
resPres (ITU9x)	Indicates whether the result, resSrc, and resDiag fields are present or not. It is coded to either PRESENT or NOT_PRESENT.
apConName (ITU9x and ANSI96)	Application context name used in a structured dialog. This field is passed through transparently by the TCAP layer and must be ASN.1 encoded/decoded by the application if the application protocol in use specifies ASN.1 encoding of this field.
result (ITU9x)	Result of a dialog section exchange. It is only present if the resPres field is set to PRESENT. 0 = TCAP_DLG_ACCEPTED Dialog has been accepted 1 = TCAP_DLG_REJ_PERM Dialog rejected permanently
resSrc (ITU9x)	Source of the result of a dialog section exchange. It is only present if the resPres field is set to PRESENT. 0x21 = TCAP_DLG_SU_TAG Dialog service user tag 0x22 = TCAP_DLG_SP_TAG Dialog service provider tag

Field	Value
resDiag (ITU9x)	Diagnostic value associated with the dialog result. It is only present if the resPres field is set to PRESENT. 0 = TCAP_DLG_RSD_NULL Result diagnostic null 1 = TCAP_DLG_RSD_NORSN No reason 2 = TCAP_DLG_RSD_NOACN User: no application context name 2 = TCAP_DLG_RSD_NCDLG 2 Provider: no common dialog portion
abortSrc (ITU9x)	Source of an aborted dialog. 0 = TCAP_DLG_USR_ABRT Service user 1 = TCAP_DLG_PRV_ABRT Service provider If a dialog is aborted by the application, this field is set to TCAP_DLG_USR_ABRT. If a dialog is aborted by the TCAP layer, this field is set to TCAP_DLG_PRV_ABRT.
secConInfo (ANSI96)	Reserved for future use.
secConInfoType (ANSI96)	Reserved for future use.
confAlgId (ANSI96)	Reserved for future use.
confAlgIdType (ANSI96)	Reserved for future use.
confValue (ANSI96)	Reserved for future use.
confValueId (ANSI96)	Reserved for future use.
apConNameType (ANSI96 only)	Specifies whether apConName is encoded as an integer or object ID. For ITU, apConName is always an object ID type.

10 Component data structures

Component structure format

The component data structure assigns components to an outgoing transaction request or receives components from an incoming transaction message. The general format of the component structure is:

```
typedef struct Tcap_Comp
{
  U8            compType;  /* Component type...                        */
  U8            rejSrc;    /* source of reject/cancel component        */
  U8            fill;      /* fill for alignment                       */
  U8            fill2;     /* fill for alignment                       */
  union                    /* ANSI or ITU-T component                  */
  {
    TcapItuComp   ituComp;   /* ITU-T component                        */
    TcapAnsiComp  ansiComp;  /* ITU-T component                        */
  } uProt;
} TcapComp;
```

Fields are coded as follows:

Field	Value
compType	Indicates the type of component and determines which protocol specific (ANSI or ITU-T) member is used: 0 = TCAP_UNKNOWN Unknown component 1 = TCAP_INVOKE Invoke 2 = TCAP_RET_RES_L Return result last 3 = TCAP_RET_ERR Return error 4 = TCAP_REJECT Reject 5 = TCAP_RET_RES_NL Return result not last 6 = TCAP_INVOKE_L Invoke last 7 = TCAP_INVOKE_NL Invoke not last 8 = TCAP_CANCEL Cancel outstanding invoke
rejSrc	Specifies the status of an incoming reject or cancel (timed out invoke) component. rejSrc distinguishes among the different reject sources: 0 = TCAP_COMP_NONE No additional status 5 = TCAP_COMP_CANCEL Invoke component timed out 6 = TCAP_COMP_REJ_USR TCAP user reject component 7 = TCAP_COMP_REJ_LOCAL Local TCAP reject component 8 = TCAP_COMP_REJ_REMOTE Remote TCAP reject component When an outstanding invoke operation times out, the TCAP layer creates a REJECT component with a rejSrc of TCAP_COMP_CANCEL. This field is not used for outgoing components.
uProt Union	Indicates protocol-specific component information specified in ANSI component structure and ITU-T component structure.

This section describes the following common data structures:

- ANSI component structure
- ITU-T component structure

ANSI component structure

The ANSI component structure is used by applications implementing ANSI TCAP:

```
typedef struct Tcap_Ansi_Comp
{
  TcapCompId      invokeId;     /* invoke id                          */
  TcapCompId      corrId;       /* correlation id                     */
  U16             invokeTimer;  /* Invoke Timer                       */
                  union         /* component type-specfic fields      */
 {
  TcapAnsiOpcode  opcode;       /* operation code for invoke          */
  TcapAnsiErrcode errcode;      /* error code for return error        */
  TcapAnsiPrbcode prbcode;      /* problem code for reject            */
} uComp;

  U8              paramFlg;     /* Set or Sequence...                 */
  U8              spare;        /* spare for alignment                */
} TcapAnsiComp;
```

Fields are coded as follows:

Field	Description
invokeId	Contains the invocation ID. It must be set for any invoke component to a value unique to all outstanding invoke components belonging to this transaction. To cancel an outstanding invoke component, set this field to the invokeID of the component that is cancelled.
corrId	Contains the optional ANSI correlation ID. • For a reply component (return result, return error, reject), set it to the invokeID of the component to which the reply is directed. • For an invoke component, it creates a linked invoke (a new invoke sent as a response to a received invoke) by setting this field to the invokeID from the original received invoke and setting the invokeId to a new unique value.
invokeTimer	Not applicable for the ANSI protocol.

opcode	Represents the operation code in an invoke component:
	`typedef struct Tcap_Ansi_Opcode`
	`{`
	`  U8  opCodeId;  /* operation code identifier */`
	`  U8  opFamily;  /* operation code family */`
	`  U8  opSpec;  /* operation code */`
	`  U8  spare;  /* spare for alignment */`
	`} TcapAnsiOpcode;`
	Refer to *TcapAnsiOpcode* on page 107 for information about the fields.

errcode	Represents the error code in a return error component:
	`typedef struct Tcap_Ansi_Errcode`
	`{`
	`  U8  errCodeId; /* error code identifier */`
	`  U8  errCode;  /* error code value */`
	`  U8  fill;  /* fill for alignment */`
	`  U8  fill2;  /* fill for alignment */`
	`} TcapAnsiErrcode;`
	Refer to *TcapAnsiErrcode* on page 109 for information about the fields.

Field	Description

prbcode	Represents the problem code in a reject component:
	`typedef struct Tcap_Ansi_Prbcode`
	`{`
	`  U8  probType;  /* problem type */`
	`  U8  probSpec;  /* problem specifier */`
	`  U8  fill;  /* fill for alignment */`
	`  U8  fill2;  /* fill for alignment */`
	`} TcapAnsiPrbcode;`
	Refer to *TcapAnsiPrbcode* on page 109 for information about the fields.

parmFlg	Specifies whether application parameters associated with this component are encoded as (ASN.1) SET or SEQUENCE: 0 = TCAP_NO_SET_SEQ No parameters 1 = TCAP_SEQUENCE Parameter is a sequence 2 = TCAP_SET Parameter is a set parmFlg is valid only for ANSI protocols. When configured with the ANSI-88 protocol, only TCAP_SET is valid. When configured for ANSI-92 or later protocols, either TCAP_SET or TCAP_SEQUENCE can be used. If parmFlg is set to TCAP_NO_SET_SEQ, then no parameter is sent, even if one is specified.

TcapAnsiOpcode

TcapAnsiOpcode contains the following fields:

Field	Value
opCodeId	Set to one of the following values: 3 = TCAP_NATIONAL National TCAP 4 = TCAP_PRIVATE Private TCAP
opFamily	There are several pre-defined fields for national TCAP. If the opCodeId specifies private TCAP, an application-specific value is used: 0x00 = TCAP_ANSI_NOF_NU Not used 0x01 = TCAP_ANSI_NOF_PARAM Parameter 0x02 = TCAP_ANSI_NOF_CHGING Charging 0x03 = TCAP_ANSI_NOF_PR_INS Provide instructions 0x04 = TCAP_ANSI_NOF_CN_CTL Connection control 0x05 = TCAP_ANSI_NOF_CL_INT Caller interaction 0x06 = TCAP_ANSI_NOF_SND_NO Send notification 0x07 = TCAP_ANSI_NOF_NET_MN Network management 0x08 = TCAP_ANSI_NOF_PROC Procedural 0x09 = TCAP_ANSI_NOF_OP_CTL Operation control family (ANSI-92) 0x0A = TCAP_ANSI_NOF_RP_EVT Report event family (ANSI-92) 0x7E = TCAP_ANSI_NOF_MISC Miscellaneous 0x7F = TCAP_ANSI_NOF_RSRVD Reserved To specify that a reply is required, the following value is logically OR'd with one of the previous values: 0x80 = TCAP_ANSI_NOF_RP_REQ Reply required OR w/value

Field	Value
opSpec	Specifies a particular operation within an operation family: 0xFF = TCAP_ANSI_OS_RSRVD All families - reserved 0x00 = TCAP_ANSI_OS_NU All families - not used 0x01 = TCAP_ANSI_OS_PRV_VAL Parameter - provide value 0x02 = TCAP_ANSI_OS_SET_VAL Parameter - set value 0x01 = TCAP_ANSI_OS_BL_CALL Charging - bill call 0x01 = TCAP_ANSI_OS_START Provide instruction - start 0x02 = TCAP_ANSI_OS_ASSIST Provide instruction assist 0x01 = TCAP_ANSI_OS_CONNECT Connection control - connect 0x02 = TCAP_ANSI_OS_TMP_CON Connection control - temporary connect 0x03 = TCAP_ANSI_OS_DISC Connection control - disconnect 0x04 = TCAP_ANSI_OS_FWD_DIS Connection control - forward disconnect 0x01 = TCAP_ANSI_OS_PLY_ANN Caller interaction - play announcement 0x02 = TCAP_ANSI_OS_COL_DIG Caller interaction - play and collect digits 0x03 = TCAP_ANSI_OS_INF_WTG Caller interaction - information waiting 0x03 = TCAP_ANSI_OS_INF_PVD Caller interaction - information provided 0x01 = TCAP_ANSI_OS_PTY_FRE Send notification when free 0x01 = TCAP_ANSI_OS_RAS_TRM Procedural report assist termination 0x01 = TCAP_ANSI_OS_CANCEL Operation control - cancel 0x01 = TCAP_ANSI_OS_VM_AVL Report event - voice message available 0x02 = TCAP_ANSI_OS_VM_RTVD Report event - voice message retrieved 0x02 = TCAP_ANSI_OS_Q_CALL Miscellaneous - queue call 0x02 = TCAP_ANSI_OS_DQ_CALL Miscellaneous - dequeue call 0x01 = TCAP_ANSI_OS_CAL_GAP Network management - call gap 0x01 = TCAP_ANSI_OS_TMP_HDN Proc. - temporary handover

TcapAnsiErrcode

TcapAnsiErrcode contains the following fields:

Field	Value
errCodeId	Set to one of the following values: 3 = TCAP_NATIONAL National TCAP 4 = TCAP_PRIVATE Private TCAP
errCode	Has several pre-defined fields for national TCAP. If the errCodeId specifies private TCAP, an application-specific value is used: 0x00 = TCAP_ANSI_ERR_NU Not used 0x01 = TCAP_ANSI_ERR_UX_CMP Unexpected component sequence 0x02 = TCAP_ANSI_ERR_UX_DAT Unexpected data value 0x03 = TCAP_ANSI_ERR_UA_NET Unavailable network resource 0x04 = TCAP_ANSI_ERR_MSG_RC Missing customer record 0x05 = TCAP_ANSI_ERR_REP_OD Reply overdue (ANSI-88) 0x05 = TCAP_ANSI_ERR_REP_SC Spare code (ANSI-92) 0x06 = TCAP_ANSI_ERR_DAT_UA Data unavailable 0x07 = TCAP_ANSI_ERR_TSK_RE Task refused (ANSI-92) 0x08 = TCAP_ANSI_ERR_Q_FULL Queue full (ANSI-92) 0x09 = TCAP_ANSI_ERR_NO_Q No queue (ANSI-92) 0x0A = TCAP_ANSI_ERR_TMR_EX Timer expired (ANSI-92) 0x0B = TCAP_ANSI_ERR_DAT_EX Data already exists (ANSI92) 0x0C = TCAP_ANSI_ERR_UNAUTH Unauthorized request (ANSI-92) 0x0D = TCAP_ANSI_ERR_NOT_QD Not queued (ANSI-92) 0x0E = TCAP_ANSI_ERR_UAS_DN Unassigned DN (ANSI-92) 0x0F = TCAP_ANSI_ERR_SPARE Spare (ANSI-92) 0x10 = TCAP_ANSI_ERR_NOT_AV Notification available on destination DN (ANSI-92) 0x11 = TCAP_ANSI_ERR_VMSR_E VMSR system ID does not match user profile (ANSI-92)

TcapAnsiPrbcode

TcapAnsiPrbcode contains the following fields:

Field	Value
probtype	Classifies the problem encountered in the rejected component: 0x00 = TCAP_ANSI_PRB_NU Not used 0x01 = TCAP_ANSI_PRB_GEN General 0x02 = TCAP_ANSI_PRB_INV Invoke 0x03 = TCAP_ANSI_PRB_RR Return result 0x04 = TCAP_ANSI_PRB_RE Return error 0x05 = TCAP_ANSI_PRB_TRANS Transaction portion 0xFF = TCAP_ANSI_PRB_RSRVD All families - reserved

Field	Value
probSpec	**General problems** 0x01 = TCAP_ANSI_PRB_UR_CMP Unrecognized component 0x02 = TCAP_ANSI_PRB_IN_CMP Incorrect component 0x03 = TCAP_ANSI_PRB_BD_CMP Badly structured component **Invoke problems** 0x01 = TCAP_ANSI_PRB_DUP_ID Duplicate invoke ID 0x02 = TCAP_ANSI_PRB_UR_OP Unrecognized operation code 0x03 = TCAP_ANSI_PRB_IN_PRM Incorrect parameter 0x04 = TCAP_ANSI_PRB_IUR_ID Unrecognized correlation ID **Return result problems** 0x01 = TCAP_ANSI_PRB_RUR_ID Unrecognized correlation ID 0x02 = TCAP_ANSI_PRB_UX_RES Unexpected return result **Return error problems** 0x01 = TCAP_ANSI_PRB_EUR_ID Unrecognized correlation ID 0x02 = TCAP_ANSI_PRB_UX_RER Unexpected return error 0x03 = TCAP_ANSI_PRB_UR_ERR Unrecognized error 0x04 = TCAP_ANSI_PRB_UX_ERR Unexpected error 0x05 = TCAP_ANSI_PRB_EN_PRM Incorrect parameter **Transaction portion problems** 0x01 = TCAP_ANSI_PRB_UR_PKG Unrecognized package type 0x02 = TCAP_ANSI_PRB_IN_TRN Incorrect transaction part 0x03 = TCAP_ANSI_PRB_BD_TRN Badly structured transaction portion 0x04 = TCAP_ANSI_PRB_UR_TRN Unrecognized transaction ID 0x05 = TCAP_ANSI_PRB_PR_TRN Permission to release 0x06 = TCAP_ANSI_PRB_RU_TRN Resource unavailable

ITU-T component structure

The ITU-T component structure is used by applications implementing ITU-T TCAP:

```
typedef struct Tcap_Itu_Comp
{
  TcapCompId       invokeId;       /* invoke id                        */
  TcapCompId       linkedId;       /* linked id                        */
  U16              invokeTimer;    /* Invoke Timer                     */
  U8               opClass;        /* operation class for invoke       */
  U8               spare;          /* spare for alignment              */
                   union           /* component type-specific fields   */
  {
    TcapItuOpcode  opcode;         /* operation code for invoke        */
    TcapItuErrcode errcode;        /* error code for return error      */
    TcapItuPrbcode prbcode;        /* problem code for reject          */
  } uComp;

} TcapItuComp;
```

Fields are coded as follows:

Field	Description
invokeID	Contains the invocation ID. It must be set for any invoke component to a value unique to all outstanding invoke components belonging to this transaction. To cancel an outstanding invoke component, this field is set to the invokeID of the component that is cancelled.
linkedId	Contains the optional ITU-T linked ID. The linked ID can be used to create a linked invoke (a new invoke sent as a response to a received invoke) by setting this field to the invokeID from the original received invoke and setting the invokeId to a new unique value.
invokeTimer	Specifies the time, in seconds, to wait for a response to an invoke component. If zero, the value from the TCAP SAP configuration is used. This field is ignored for any component other than an invoke component.
opClass	Specifies the ITU-T operation class. It is encoded to one of the values specified in the opClass table.

opcode	Represents the operation code in an invoke component:
	`typedef struct Tcap_Itu_Opcode`
	`{`
	`  U8  opCodeType;  /* operation code type */`
	`  U8  spare;  /* spare for alignment */`
	`  U8  fill;  /* fill for alignment */`
	`  U8  fill2;  /* fill for alignment */`
	`  TcapOctetStr opCode;  /* opcode length/value */`
	`} TcapItuOpcode;`
	The opCode value is a variable length octet string coded to the standards for the application protocol in use.
	opCodeType can be set to one of the following values:
	0 = TCAP_NONE Opcode omitted (return result only)
	1 = TCAP_LOCAL Local error/operation code
	2 = TCAP_GLOBAL Global error/operation code

Field	Description

errcode	Represents the error code in a return error component:
	`typedef struct Tcap_Itu_Errcode`
	`{`
	`  U8  errCodeId;  /* error code identifier */`
	`  U8  fill;  /* fill for alignment */`
	`  U8  fill2;  /* fill for alignment */`
	`  U8  spare;  /* spare for alignment */`
	`  TcapOctetStr errCode;  /* error code length/value */`
	`} TcapItuErrcode;`
	
	The errCode value is a variable length octet string coded to the standards for the application protocol in use. errCodeId can be set to either:
	1 = TCAP_LOCAL Local error/operation code
	2 = TCAP_GLOBAL Global error/operation code

prbcode	Represents the problem code in a reject component:
	`typedef struct Tcap_Itu_Prbcode`
	`{`
	`  U8  probType;  /* problem type */`
	`  U8  spare;  /* spare for alignment */`
	`  U8  fill;  /* fill for alignment */`
	`  U8  fill2;  /* fill for alignment */`
	`  TcapOctetStr prbCode;  /* problem code length/value */`
	`} TcapItuPrbcode;`
	
	The probType field classifies the problem encountered in the rejected component:
	0x00 = TCAP_PROB_NONE No problem code flag
	0x80 = TCAP_PROB_GENERAL General problem code flag
	0x81 = TCAP_PROB_INVOKE Invoke problem code flag
	0x82 = TCAP_PROB_RET_RES Return result problem code
	0x83 = TCAP_PROB_RET_ERR Return error problem code flag
	
	The prbCode value is a variable length octet string:
	General problems
	0 = TCAP_UNREC_COMP Unrecognized component
	1 = TCAP_MISTYPED_COMP Mistyped parameter
	2 = TCAP_BAD_STRUC_COMP Badly structured component
	Invoke problems
	0 = TCAP_DUP_INVOKE Duplicate invoke ID
	1 = TCAP_UNREC_OPR Unrecognized invoke ID
	2 = TCAP_MISTYPED_PARAM Mistyped parameter
	3 = TCAP_RESOURCE_LIMIT Resource limitation
	4 = TCAP_INIT_RELEASE Initiating release
	5 = TCAP_UNREC_LINKED_ID Unrecognized linked ID
	6 = TCAP_LINKED_RESP_UNX Linked response unexpected
	7 = TCAP_UNX_LINKED_OP Unexpected linked operation
	Return result problem
	0 = TCAP_RR_UNREC_INVKID Unrecognized invoke ID
	1 = TCAP_UNX_RETRSLT Return result unexpected
	2 = TCAP_RR_MISTYPED_PAR Mistyped parameter
	Return error problems
	0 = TCAP_RE_UNREC_INVKID Unrecognized invoke ID
	1 = TCAP_RE_UNX_RETERR Unexpected return error
	2 = TCAP_UNREC_ERROR Unrecognized error
	3 = TCAP_UNX_ERR Unexpected error
	4 = TCAP_RE_MISTYPED_PAR Mistyped parameter

The opclass field can be one of the following values:

Value	Description
1	Both successes and failures are reported. A timeout is an abnormal failure.
2	Only failure is reported. A timeout implies successful completion.
3	Only success is reported. A timeout implies a failed operation.
4	Neither success or failure is reported. There is no interpretation of timeout.

11 Incoming message event structures

Messages overview

TCAPRetrieveMessage is unique in that the event structure associated with a received message depends on the type of message received from the TCAP layer. This section provides a description of each possible event, as well as the structures that are returned with each event.

General receive information block structure

Describes incoming events to the calling application. It contains an event type and a service provider ID that are common to all received events and an event-specific section:

```
typedef struct Tcap_Recv_Info
{
  U8                  eventType; /* Event type...                    */
  U8                  spare;     /* spare for alignment              */
  S16                 suId;      /* TCAP SAP ID event belongs to     */
                      union      /*  event-specific info             */
  {
    TcapCoordEvent    coord;     /* coord indication/confirm         */
    TcapPCStateEvent  pcstate;   /* SP state change                  */
    TcapSsnStateEvent ssnstate;  /* subsystem state change           */
    TcapNotifEvent    notif;     /* undeliverable msg notify         */
    TcapStatusEvent   status;    /* TCAP error indication            */
    TcapTransEvent;   data;      /* transaction data                 */
  } event;

} TcapRecvInfo;
```

The TcapRecvInfo structure contains the following fields:

Field	Description
eventType	Identifies the event received, and determines which event-specific union member is used. 0xB1 = TCAP_EVENT_DAT_IND Transaction data indicator 0xB2 = TCAP_EVENT_STA_IND Status (error) indicator 0xB3 = TCAP_EVENT_COORD_IND Coordinated state change indication 0xB4 = TCAP_EVENT_COORD_CFM Coordinated state change confirmation 0xB5 = TCAP_EVENT_SSN_STATE Subsystem state change 0xB6 = TCAP_EVENT_PC_STATE SP state change 0xBF = TCAP_EVENT_NOT_IND Notice indication (SCCP undeliverable message return)
suID	Identifies the TCAP SAP to which the incoming message belongs. This is useful to applications that bind to more than one TCAP SAP, such as ones that implement multiple subsystems in a single application.

TCAP coordinated event structure

Represents both incoming coordinated status change indication (remote request) and confirmation (response) events. It is indicated by an eventType of TCAP_EVENT_COORD_IND or TCAP_EVENT_COORD_CFM.

```
typedef struct Tcap_Coord_Event
{
  U8   aSsn;   /* affected subsystem number              */
  U8   smi;    /* subsystem multiplicity indicator...    */
  U8   fill;   /* fill for alignment                     */
  U8   fill2;  /* fill for alignment                     */
} TcapCoordEvent;
```

The TcapCoordEvent structure contains the following fields:

Field	Description
aSsn	Represents the affected subsystem number. The value can be in the range of 0 - 255.
smi	Represents the subsystem multiplicity indicator, or the status of the replicated subsystem from the underlying SCCP message: 0x00 = SMI_UNKNOWN Multiplicity unknown 0x01 = SMI_SOLO Subsystem not replicated 0x02 = SMI_DUP Subsystem is replicated 0x10 = SMI_DENIED Indicate denial of coordinated state change SMI_DENIED is used by the SCCP in the TCAP_EVENT_COORD_CFM message to indicate that the state change request timed out and was not granted.

Signaling point status event structure

Notifies the application of a change in the state of a concerned signaling point (eventType of TCAP_EVENT_PC_STATE).

```
typedef struct Tcap_PCState_Event
{
  U32  aPc;          /* affected point code           */
  U8   status;       /* new signaling point status    */
  U8   spare;        /* spare for alignment           */
  U8   fill;         /* fill for alignment            */
  U8   fill2;        /* fill for alignment            */
U32  opc;           /* originating point code        */
} TcapPCStateEvent;
```

The TcapPCStateEvent structure contains the following fields:

Field	Description
aPc	Contains the point code of the affected signaling point.
status	Represents the new status of the affected signaling point: 0x00 = SP_ACC Signaling point accessible 0x01 = SP_INACC Signaling point inaccessible 0x06 = SP_INACC_NODROP Signaling point inaccessible - connections not dropped 0x07 = SP_CONG_OFF APC congestion ceased 0x10 = SP_CONG1 APC congestion level 1 0x11 = SP_CONG2 APC congestion level 2 0x12 = SP_CONG3 APC congestion level 3
opc	Originating point code - used for multiple OPC support.

Subsystem status event structure

Notifies the application of a change in the state of a subsystem at a concerned signaling point (eventType of TCAP_EVENT_SSN_STATE):

```
typedef struct Tcap_Ssn_State_Event
{
  U8   aSsn;    /* affected subsystem number                          */
  U8   status;  /* new subsystem status                               */
  U8   smi;     /* subsystem multiplicity indicator                   */
  U8   spare;   /* spare for alignment                                */
  U32  aPc;     /* affected point code - future use                   */
  U32  opc;     /* originating point code                             */
} TcapSsnStateEvent;
```

The TcapSsnStateEvent structure contains the following fields:

Field	Description
aSsn	Contains the affected subsystem number. Valid range is 0 - 255.
status	Represents the new status of the affected subsystem: 0x03 = SS_OOS Subsystem out of service 0x04 = SS_IS Subsystem in service
smi	Represents the subsystem multiplicity indicator, or the new status of the subsystem with respect to replication: 0x00 = SMI_UNKNOWN Multiplicity unknown 0x01 = SMI_SOLO Subsystem not replicated 0x02 = SMI_DUP Subsystem is replicated
aPc	Not currently available. Reserved for future use.
opc	Originating point code - used for multiple OPC support.

TCAP notification event structure

Notifies an application that a message could not be delivered by the SCCP layer (eventType of TCAP_EVENT_NOT_IND).

```
typedef struct Tcap_Notif_Event
{
  TcapTransInfo  transInfo;    /* trans. info from failed Req          */
  U8             retcause;     /* cause for return of message          */
  U8             spare;        /* spare for alignment                  */
  U8             fill;         /* fill for alignment                   */
  U8             fill2;        /* fill for alignment                   */
} TcapNotifEvent;
```

The TcapNotifEvent structure contains the following fields:

Field	Description
transInfo	Contains the message type, dialog IDs, and SCCP addresses from the failed request.
retcause	Cause for the return of the message. This field must be set to one of the following values: 0x00 = RETCGENTRANS No translation for address of this nature 0x01 = RETCNOTRANS No translation for this address 0x02 = RETCSUBSCONG Subsystem congestion 0x03 = RETCSUBSFAIL Subsystem failure 0x04 = RETCUNQUIP Unequipped user 0x05 = RETCNETFAIL Network failure 0x06 = RETCNETCONG Network congestion 0x07 = RETCUNQUALIFIED Unqualified 0x08 = RETCHOPCNT Hop counter violation 0x09 = RETCMSGXPORT Error in message transport 0x0A = RETCLOCALPROC Error in local processing 0x0B = RETCREASSEMBLY Destination cannot do re-assembly 0xF9 = RETCBADISNI Invalid ISNI routing request 0xFA = RETCAUTH Unauthorized message 0xFB = RETCINCOMPAT Message incompatibility 0xFC = RETCNOISNI Cannot do ISNI constrained routing 0xFD = RETCREDISNI Redundant ISNI constrained routing information 0xFE = RETCISNIID Cannot do ISNI identification

TCAP status event structure

Notifies the application that it could not correctly process the request (eventType of TCAP_EVENT_STA_IND). It is usually accompanied by an alarm that indicates the cause of the failure. This event is primarily intended as a debugging aid during system development and integration.

```
typedef struct Tcap_Status_Event
{
  S16  status;   /* cause of error           */
  S16  suId      /* caller's service user ID */
} TcapStatusEvent;
```

The TcapStatusEvent structure contains the following fields:

Field	Description
status	Identifies that cause of the error: 1 = TCAP_R_EVT_INAPP Received event in inappropriate state 2 = TCAP_MSNG_ELE Missing mandatory element 3 = TCAP_DUP_INV_ID Duplicate invoke ID 4 = TCAP_INV_REJ Received reject with syntax error 5 = TCAP_CONGESTED Unable to initiate new transaction due to outbound congestion 6 = TCAP_RES_FAIL Unable to initiate new transaction due to dialog control block exhaustion
suId	Contains the calling application service user ID from the bind request.

TCAP transaction data event structure

Represents an incoming transaction message (eventType of TCAP_EVENT_DAT_IND):

```
typedef struct Tcap_Trans_Event
{
  U32            opc;         /* originating point code from
                               * routing label                        */
  TcapTransInfo  transInfo;   /* transaction info block               */
  TcapDlgSect    dlgPart;     /* ITU-92 dialog section                */
  U16            numComps;    /* number of components in msg          */
  U16            userInfoLen; /* byte length of user info             */
  U8             *pUserInfo;  /* pointer to user dialog info          */

} TcapTransEvent;
```

The TcapTransEvent structure contains the following fields:

Field	Description
opc	Originating point code from the routing label of the incoming message.
transInfo	Contains the transaction message type, dialog IDs, and originating and destination SCCP addresses.
dlgPart	Contains the dialog portion (application context name, etc.) of the message. It is used only for ITU-92 or later and ANSI-96 and later protocols.
numComps	Contains a count of the number of components in the received message.
userInfoLen	Contains the number of bytes in the user information section of the dialog portion message. It is used only for ITU-92 or later and ANSI-96 and later protocols. It is set to zero if there was no user information in the dialog portion of the message and for protocol variants other than ITU-92 or later and ANSI-96 and later protocols.
pUserInfo	Contains the address of the user information section of the dialog portion message. It is used only for ITU-92 or later and ANSI-96 and later protocols. It is set to NULL if there was no user information in the dialog portion of the message and for protocol variants other than ITU-92 or later and ANSI-96 and later protocols.

Index

8

800 number client 86

800 number server 86

A

abnormal conditions 44

address override 31

addressing and routing 31

ANSI component structure 106

ANSI transaction types 22

application programming models 18

B

backup subsystem status 50

C

checkpointing 22

common data structures 97, 98, 100, 101, 103

component data structures 105, 106, 111

configuration 30, 90, 91

congestion 26

contexts and queues 18

conversational linked transaction 42

D

data types 93

demonstration programs 85

dialog IDs 36

E

entity 16

events 39, 115, 115, 116, 117, 118, 119, 120, 121

F

find800 86

functions 16, 53, 54, 67, 68

G

General receive information block structure 115

global title translation 31, 33, 94

I

inactivity timeout 45

inbound congestion 29

instance ID 16

invalid components 44

invoke time-outs 44

ITU-T component structure 111

ITU-T transaction types 22

M

management functions 16, 67, 68

message length 24

N

Natural Access 18, 37

network overload 26

O

octet strings 93

outbound congestion 26

out-of-service 50

P

package types 22

point codes 93

programming model 15

Q

QOS 21, 100

quality of service 21, 100

queues and contexts 18

R

redundancy 40

remote signaling points 51

request and response transaction find800

routing 31

S

SAP 15

SCCP 21, 31, 33

SCCP address override 31

SCCP address structure 98

SCCP quality of service (QOS) structure 100

SccpAddr 98

segmentation 24

service access points 15

service functions 16, 53, 54

signaling point 50

Signaling point status event structure 117

simple transaction 41

SS7 architecture 11

state changes 50

status and notify indications 36

subsystem status 50

Subsystem status event structure 118

T

TCAP coordinated event structure 116

TCAP dialog section structure 103

TCAP notification event structure 119

TCAP status event structure 120

TCAP task 13

TCAP transaction data event structure 121

TCAP transaction information structure 101

TCAPAddComp 55

TCAPAlarmControl 69

TcapAnsiComp 106

TcapAnsiErrcode 109

TcapAnsiOpcode 107

TcapAnsiPrbcode 109

TCAPAPISTATS 58

tcapcfg 90

TcapComp 105

TcapCompId 94

TcapCoordEvent 116

TCAPCoordReq 56

TCAPCoordResp 57

TcapDlgSect 103

TCAPGenCfg 70, 74

TCAPGenStatus 71

TCAPGetApiStats 58

TCAPGetComp 59

TCAPGetGenCfg 72

TCAPGetSapCfg 73

TCAPInitGenCfg 74

TCAPInitMgmtAPI 76

TCAPInitSapCfg 77

TCAPInitTrans 61

TcapItuComp 111

TcapNotifEvent 119

TcapOctetStr 93

TcapPCStateEvent 117

TcapRecvInfo 115

TCAPRetainTrans 62

TCAPRetrieveMessage 63, 115

TCAPSapCfg 77, 80

TCAPSapStats 81

TcapSccpQos 100

TcapSsnStateEvent 118

TCAPStateReq 64

TcapStatusEvent 120

TCAPTermMgmtAPI 83

TCAPTraceControl 84

TcapTransEvent 121

TcapTransInfo 101

TCAPTransRqst 65
tracing 52
transactions 22, 40, 44

U

unconfirmed operations 16
utilities 85