Dialogic
Making Innovation Thrive

Dialogic® TX Series SS7 Boards

Health Management Developer's Reference Manual

July 2009 64-0455-01

www.dialogic.com

Copyright and legal notices

All contents of this document are furnished for informational use only and are subject to change without notice and do not represent a commitment on the part of Dialogic Corporation or its subsidiaries ("Dialogic"). Reasonable effort is made to ensure the accuracy of the information contained in the document. However, Dialogic does not warrant the accuracy of this information and cannot accept responsibility for errors, inaccuracies or omissions that may be contained in this document.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH DIALOGIC® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN A SIGNED AGREEMENT BETWEEN YOU AND DIALOGIC, DIALOGIC ASSUMES NO LIABILITY WHATSOEVER, AND DIALOGIC DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF DIALOGIC PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHT OF A THIRD PARTY.

Dialogic products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Due to differing national regulations and approval requirements, certain Dialogic products may be suitable for use only in specific countries, and thus may not function properly in other countries. You are responsible for ensuring that your use of such products occurs only in the countries where such use is suitable. For information on specific products, contact Dialogic Corporation at the address indicated below or on the web at www.dialogic.com.

It is possible that the use or implementation of any one of the concepts, applications, or ideas described in this document, in marketing collateral produced by or on web pages maintained by Dialogic may infringe one or more patents or other intellectual property rights owned by third parties. Dialogic does not provide any intellectual property licenses with the sale of Dialogic products other than a license to use such product in accordance with intellectual property owned or validly licensed by Dialogic and no such licenses are provided except pursuant to a signed agreement with Dialogic. More detailed information about such intellectual property is available from Dialogic's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. Dialogic encourages all users of its products to procure all necessary intellectual property licenses required to implement any concepts or applications and does not condone or encourage any intellectual property infringement and disclaims any responsibility related thereto. These intellectual property licenses may differ from country to country and it is the responsibility of those who develop the concepts or applications to be aware of and comply with different national license requirements.

Dialogic, Dialogic Pro, Brooktrout, Diva, Cantata, SnowShore, Eicon, Eicon Networks, NMS Communications, NMS (stylized), Eiconcard, SIPcontrol, Diva ISDN, TruFax, Exnet, EXS, SwitchKit, N20, Making Innovation Thrive, Connecting to Growth, Video is the New Voice, Fusion, Vision, PacketMedia, NaturalAccess, NaturalCallControl, NaturalConference, NaturalFax and Shiva, among others as well as related logos, are either registered trademarks or trademarks of Dialogic Corporation or its subsidiaries. Dialogic's trademarks may be used publicly only with permission from Dialogic. Such permission may only be granted by Dialogic's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. Any authorized use of Dialogic's trademarks will be subject to full respect of the trademark guidelines published by Dialogic from time to time and any use of Dialogic's trademarks requires proper acknowledgement.

Windows is a registered trademark of Microsoft Corporation in the United States and/or other countries. Other names of actual companies and product mentioned herein are the trademarks of their respective owners.

This document discusses one or more open source products, systems and/or releases. Dialogic is not responsible for your decision to use open source in connection with Dialogic products (including without limitation those referred to herein), nor is Dialogic responsible for any present or future effects such usage might have, including without limitation effects on your products, your business, or your intellectual property rights.

Using the AMR-NB resource in connection with one or more Dialogic products mentioned herein does not grant the right to practice the AMR-NB standard. To seek a patent license agreement to practice the standard, contact the VoiceAge Corporation at http://www.voiceage.com/licensing.php.

Revision history

Revision	Release date	Notes
1.0	May, 1999	GJG
1.1	April, 2000	GJG, SS7 3.5
1.2	November, 2000	GJG, SS7 3.6
1.3	August, 2001	GJG, SS7 3.8 Beta
1.4	February, 2002	MVH, SS7 3.8 GA
1.5	November, 2003	LBG/SRG, SS7 4.0 Beta
1.6	April, 2004	LBG, SS7 4.0
1.7	October, 2008	SRG/DEH, SS7 5.0
64-0455-01	July 2009	LBG, SS7 5.1
Last modified: July 7, 2009		

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

Table Of Contents

1 Introduction

The *Dialogic® TX Series SS7 Boards Health Management Developer's Reference Manual* explains how to:

- Develop and set up redundant applications
- Detect and recover from system failures
- Use the Health Management service

This manual defines telephony terms where applicable, but assumes that the reader is familiar with basic telephony and Internet data communication concepts, switching, and the C programming language.

Note: The product to which this document pertains is part of the NMS Communications Platforms business that was sold by NMS Communications Corporation ("NMS") to Dialogic Corporation ("Dialogic") on December 8, 2008. Accordingly, certain terminology relating to the product has been changed. Below is a table indicating both terminology that was formerly associated with the product, as well as the new terminology by which the product is now known. This document is being published during a transition period; therefore, it may be that some of the former terminology will appear within the document, in which case the former terminology should be equated to the new terminology, and vice versa.

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

2 Health management

System overview

The Health Management system is a set of hardware and software components that supports SS7 redundancy and the development of distributed, highly available call processing systems that employ SS7 signaling. These systems can detect and recover from signaling link failures, board failures, and node failures without a total service outage. The Health Management architecture facilitates the design of systems whose hardware or software components can be upgraded, or whose call-handling capacity can be increased or decreased.

The core of the architecture is an extended SS7 software capability that allows two TX boards to be paired in a primary or backup arrangement. The boards are connected by a private high speed Ethernet link that allows them to exchange heartbeats, signaling messages, and state information.

The TX boards can be spread across two signaling nodes (multiple chassis) or be located in the same signaling node (single chassis). The two boards appear to the rest of the SS7 network as a single signaling point (SP) with a single point code. In a SIGTRAN configuration, the two boards appear as a single point code, but each board has a separate IP address.

The Health Management system can also be used in a non-redundant single board configuration, known as a standalone configuration, to monitor and control the board.

SS7 layers

The following table describes how the SS7 layers work in a redundant configuration:

Applicable configurations	SS7 layer	Description
All redundant configurations except SIGTRAN	MTP 2	Active on both boards, allowing all configured signaling links to be active and eliminating the need for provisioning of spare signaling links.
	MTP 3	MTP 3 routing and management functions are operational only on the primary board. Link and route status changes are checkpointed to the backup MTP 3 layer to ensure that it has up-to-date network status information in case of a primary board outage.
All redundant configurations	ISUP	Operates in a primary and backup mode, with all circuit switched connections managed by the active board. Call state information can be checkpointed by the local application to the backup ISUP entity, through extensions to the normal call processing APIs, so that stable calls can be preserved across a signaling board or node outage.
	TUP	Preserves stable calls with automated checkpoints between the primary and backup tasks. The primary and backup application must also checkpoint call states to facilitate smooth switchovers.
	SCCP	Operates along with other SS7 layers in a two-board redundant primary and backup configuration, in addition to the current single-board standalone configuration. The objective of the redundant configuration is to maintain the SCCP service across a failure. The backup SCCP layer can re-synchronize its internal state with the primary SCCP layer in cases where communication with the primary board is lost and then re-established, or when the backup board has been reloaded due to a failure or to routine maintenance.
	TCAP	Operates in a primary and backup mode. To allow a backup TCAP task to immediately take over service, the primary TCAP task sends checkpoint messages to inform the backup task of changes in various TCAP transactions. Additionally, if the primary and backup tasks become disconnected due to a failure or a reloading of the backup board, the backup task retrieves the current transaction states for all of the transactions on the primary task.

SIGTRAN layers

In a SIGTRAN configuration, the private Ethernet between the mate boards is not used by SCTP or M3UA for data or checkpoint messages. The private Ethernet is required for TXMON heartbeat messages and higher layer checkpointing.

Both the primary and backup boards establish associations with the remote endpoints when the boards start up. The association from the backup board remains in a stand-by state until the primary board fails or a planned switchover occurs. No data is passed over the association from the backup board until that board becomes the primary.

Signaling

Operation of the signaling subsystem is under complete control of the local signaling application. The application designates each board as either the primary or backup board after it is downloaded. During normal operation, applications using SCCP behave normally. There are no checkpointing responsibilities, other than updating a backup host in the dual chassis arrangement (if necessary). For class 0 connectionless service, best effort delivery service is maintained across switchovers. No other state information, other than the accessible or inaccessible status of the remote SP/SSN, is maintained between primary and backup SCCP layers.

For class 1 connectionless service, SLS values assigned to a sequence are not retained across switchovers. No checkpointing of SLS assignments (SCLI data structures) is required. The backup must, however, avoid re-using frozen segmentation local references (those recently assigned by the primary) for some period after a switchover, so their usage must be synchronized with the backup application.

In general, for both classes of connectionless service, messages can be lost on a switchover. Any detection and recovery of lost messages is the responsibility of the application-level protocol running above SCCP.

For both classes of service, segmented messages in the process of being transmitted or received are lost or discarded on a switchover. If the remaining segments of a partially reassembled incoming message that was lost or discarded due to a switchover are received by the (new) primary, they are detected and discarded. If any of these segments has the return option set, it is returned to the sender in an XUDTS message with a return cause of segmentation failed for ITU or error in message transport for ANSI.

During normal operation, applications using TCAP behave normally. TCAP transaction information is checkpointed by the primary TCAP task and is configurable. An application can configure each user SAP to, by default, checkpoint all transactions, checkpoint only those initiated by the application, or checkpoint no transactions. The default checkpoint action can be overridden by an application, which can checkpoint transactions on an individual basis.

A transaction can be checkpointed at any time during the transaction lifetime. For example, after a begin message is received, the application sends a continue message and specifies that the transaction must be checkpointed. Although the begin message was not checkpointed, the transaction is checkpointed as the continue message is sent. The TCAP task keeps track of which transactions are checkpointed and deletes the checkpoints as the transactions are closed.

If using ISUP, the application must checkpoint call status changes to the ISUP layer on the backup board, as necessary to preserve stable calls. Upon detection of a failure of the primary signaling board (through the Health Management system) or failure of the primary application or signaling node (through application-specific means), the application directs the backup signaling board to become the primary board and take over signaling operations. When a failed signaling board is restored to service as the backup, the application can re-synchronize it with the primary board by checkpointing the state of each circuit through the call processing extensions.

If using TUP, call states are synchronized automatically between the two TX boards. The applications must do the same.

System requirements

Health Management is supported on all TX boards.

Software

The Health Management system is composed of the following software:

Software	Description
txmon	Executes on the TX board, monitoring its mate and the SS7 tasks. *txmon* signals to a host process every 200 milliseconds. Heartbeats contain the current run state (for example, primary, backup, task failure) of the board and the current link state (connected or isolated) of the IBC link.
Health Management Interface (HMI) service	A daemon process running on the host that constantly monitors the status of the boards. Watches the heartbeats for state changes and for the absence of the heartbeats. Any change of either state generates an asynchronous event to all registered applications. For information on installing the HMI service, refer to *Board installation and cabling* on page 73.
Health Management service (HM API)	A Natural Access service that allows applications to control the behavior of the system and to monitor the performance of the Health Management system. Many events are generated to registered applications. For example, a separate task failure (HMI_EVN_TASKDEAD) event is passed that is different from the event generated by the absence of heartbeats (HMI_EVN_BRDDEAD). An application can ask the HMI service for a status to determine which task failed when the HMI_EVN_TASKDEAD event is generated. An application can also control the behavior of the Health Management system using **hmiPrimary** and **hmiBackup**.

3 Health Management programming model

Programming model overview

The Health Management system allows applications (call processing, user interface, or OAM applications) to perform certain requests to control the operation of the signaling subsystem, as well as monitor for unsolicited system status events. Separate connections, or handles, are used for the two types of operations.

The Health Management system supports user-supplied call processing applications that monitor the system status and take appropriate actions, and user interface applications that display the current system status and statistics as well as initiate switchovers and resets.

The Health Management system does not distinguish between these two types of applications. All registered applications can receive unsolicited status events and can issue any of the supported operation requests. There are a maximum number of connections (for either type of operation) to the Health Management Interface (HMI) service available to all the applications. HMI supports up to 16 simultaneous application connections.

Unsolicited status events

To register for unsolicited status events, the application calls **ctaOpenServices**. This call creates a connection to the Health Management Interface service and returns a Natural Access handle that can be used to wait for incoming messages (**ctaWaitEvent**). When **ctaWaitEvent** returns or completes, the application has received an event on which to possibly take action. To disconnect from HMI (such as when the application shuts down), the application calls **ctaCloseServices** to free up the connection slot for other applications.

HMI reports the current run state of the boards for which an application registers. For example, if a board is in the primary state, the HMI_EVN_PRIMARY event is generated to the registering application.

Application requests

Status, statistics, and control requests are comprised of messages exchanged between the calling application and the HMI. These functions require a separate connection to the HMI and operate in a blocking fashion, similar to remote procedure calls. The function sends a request to the HMI and waits for the response message (up to 15 seconds), blocking the calling process or thread.

The application calls **ctaOpenServices** to establish a connection to the HMI. Once the connection is opened, the application can call any of the other Health Management functions. Each function request generates a message to the HMI and waits for the response message - either a confirm response for a successful operation or a refuse response for an unsuccessful operation. The application terminates the connection to the HMI by calling **ctaCloseServices**.

For some request functions, it may take several seconds to receive a response message. For some applications, blocking for this length of time is not appropriate. These applications must spawn a separate thread to perform the function call, ensuring that the main or worker threads are not blocked for an extended period of time.

Sharing the handle returned from **ctaOpenServices** among several threads, each of which might generate independent requests, is not recommended, as the response messages can get mixed in among the requesting threads. In this case, each thread must perform its own **ctaOpenServices** and use its own handle.

Status and statistics requests are acceptable in any state. The appropriate response message is returned on the connection between the HMI and the application.

Reset requests instruct the HMI to re-read its configuration file. The request is primarily intended to provide an operational means to address unplanned conditions or configuration updates. The primary or backup state information is not reset at this point.

ctaOpenServices

The Health Management service passes parameters to **ctaOpenServices** in the svcargs.args element of the CTA_SERVICE_DESC structure, as described in the following table:

Element	Description
args[0]	A unique index for each connection made to the Health Management service, 0 - 15. There is a maximum of 16 open connections.
args[1]	Identifies which board this channel manages. If this is a call to open an event channel, this can be set to HMI_EVENTS_ALL_BOARDS to receive events for all boards managed by the HMI service.
args[2]	Set to HMI_RCV_EVENTS to receive events or set to HMI_DO_COMMANDS to perform actions.

The Natural Access service name is *hmi* and the Natural Access service manager name is *hmimgr*. These names should be placed in the CTA_SERVICE_DESC structure when opening the service. These names can also be edited into the *cta.cfg* file to facilitate using the tracing service, as shown in the following sample code:

```
[ctasys]
Service = adi, adimgr
Service = hmi, hmimgr    # trace the HM API

TraceMask = 0
StartWebServer = 1       # Change to 0 to disable ctdaemon web server.
StartTraceServer = 1     # Change to 0 to disable ctdaemon trace server.
HttpPort  = 1100         # TCP/IP port for web server.
TracePort = 1101         # TCP/IP port for trace server.
TraceMaxControllers = 1  # Num. clients allowed to set tracemask.
TraceMaxMonitors = 10    # Num. clients allowed to monitor trace msgs.
[ctapar]
[eof]
```

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

Events

An application can register for asynchronous events. The registering application can request all events for all configured TX boards or it can register for specific boards, one at a time. The event and board number are the only relevant elements in events received by the application, in the event and value fields of the Natural Access CTA_EVENT structure. The following table lists the possible events:

Event	Description
HMI_EVN_BRDDEAD	Board is dead, must reload.
HMI_EVN_CONFLICT	There is confusion in the state.
HMI_EVN_CONNECTED	Mate board is now available from this board.
HMI_EVN_EXTRACT	TX board extraction is pending.
HMI_EVN_HALTED	Previously requested halt is finished.
HMI_EVN_INSERT	TX board was inserted.
HMI_EVN_ISOLATED	Mate board is now unavailable from this board.
HMI_EVN_LOADING	Previously requested load has started.
HMI_EVN_NETWORK_DOWN	Board lost SIGTRAN connectivity with all configured remote nodes.
HMI_EVN_NETWORK_UP	Board gained SIGTRAN connectivity to one or more configured remote nodes.
HMI_EVN_NOWBACKUP	Node is now the backup node.
HMI_EVN_NOWPRIMARY	Node is now the primary node but the application layers may not yet be available.
HMI_EVN_NOWSTANDALONE	Node is now stand alone.
HMI_EVN_SERVICE_DOWN	Node is shutting down. All connections must be closed. The board number is not relevant for this event.
HMI_EVN_STARTING	Node is freshly started.
HMI_EVN_STOP	Application must close communications to the TX board (close services).
HMI_EVN_TASKDEAD	A task on the board is dead, must reload.

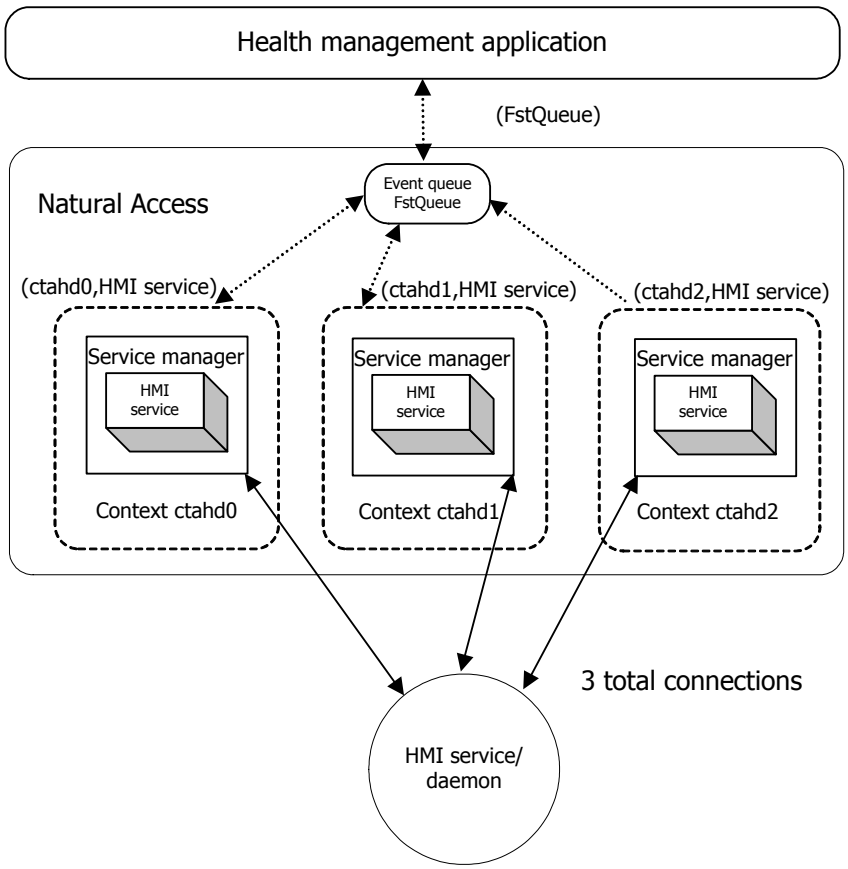
Sample setup

The following code sample opens the service for TX board ***boardNum*** and then registers for events from that boards. In this sample, after opening these services, a loop watching for events prints out the data from the events as they are received.

```
/* Service name/manager pair for ctaInitialize */
CTA_SERVICE_NAME    HmiServiceNames[] = { { "HMI", "HMIMGR" } };

/* Service list for ctaOpenServices */
CTA_SERVICE_DESC    HmiOpenSvcLst[] =
{
    {{"HMI", "HMIMGR"}, {0}, {0}, {0}}
};

hmiInitparms.size = sizeof( CTA_INIT_PARMS );
hmiInitparms.parmflags = CTA_PARM_MGMT_SHARED;
hmiInitparms.ctacompatlevel = CTA_COMPATLEVEL;
ret = ctaInitialize( HmiServiceNames, 1, &hmiInitparms );

/* create a CT Access queue used to pass all events to the application */
ret = ctaCreateQueue( NULL, 0, &AppCtlQueue );

/* create a CT Access context for HMI async events
 * and open the HMI async events service on that context
 */
ret = ctaCreateContext( AppCtlQueue, 0, "APPCTL", &AppCtlAsyncHd );

HmiOpenSvcLst[0].svcargs.args[0] = hmiCtlChan;
HmiOpenSvcLst[0].svcargs.args[1] = boardNum;
HmiOpenSvcLst[0].svcargs.args[2] = HMI_RCV_EVENTS;
ret = ctaOpenServices( &AppCtlAsyncHd, &HmiOpenSvcLst[0], 1 );

/* Wait for service open to complete. */
do
{
    ctaWaitEvent( AppCtlQueue, &event, CTA_WAIT_FOREVER );
} while (event.id != CTAEVN_OPEN_SERVICES_DONE);

if (event.value != CTA_REASON_FINISHED)
{
    ctaGetText( event.ctahd, event.value, emsg, sizeof( emsg ) );
    printf( "ERROR opening HMI service [%ld]: %s\n", hmiCtlChan, emsg );
    exit( 1 );
}

/* create a CT Access context for HMI commands
 * and open the HMI commands service on that context
 */
ret = ctaCreateContext( AppCtlQueue, 0, "APPCMD", &AppCtlCmdHd );

HmiOpenSvcLst[0].svcargs.args[0] = hmiCmdChan;
HmiOpenSvcLst[0].svcargs.args[1] = boardNum;
HmiOpenSvcLst[0].svcargs.args[2] = HMI_RCV_EVENTS;
ret = ctaOpenServices( &AppCtlCmdHd, &HmiOpenSvcLst[0], 1 );

/* Wait for service open to complete. */
do
{
    ctaWaitEvent( AppCtlQueue, &event, CTA_WAIT_FOREVER );
} while (event.id != CTAEVN_OPEN_SERVICES_DONE);

if (event.value != CTA_REASON_FINISHED)
{
    ctaGetText( event.ctahd, event.value, emsg, sizeof( emsg ) );
    printf( "ERROR opening HMI service [%ld]: %s\n", hmiCmdChan, emsg );
    exit( 1 );
}
```

```
while (!stop)
{
    ret = ctaWaitEvent( AppCtlQueue, &event, CTA_WAIT_FOREVER );
    if (ret == SUCCESS)
    {
        printf( "Received event %x from board %d\n", event.id, event.value );
        if (event.id == HMI_EVN_SERVICE_DOWN)
            stop = TRUE;
    }
    else
    {
        ctaGetText( NULL_CTAHD, ret, emsg, sizeof( emsg ) );
        printf( "ERROR waiting on CTA event: %s\n", emsg );
        stop = TRUE;
    }
}
```

The application opens the number of HMI service interfaces needed to monitor and control health management. The following illustration shows the arrangement between the Natural Access queues and contexts when three opens were performed:



In a single node system, an application can control both boards by opening HMI services to each board. This application can take appropriate actions to recover from board failures (for example, by making backup boards primary, reloading dead boards). In a dual node system, the applications are responsible for choreographing failovers and switchovers. Consider these situations when designing the user part application. Refer to *ISUP demonstration program overview* on page 99 for an application that can be run in dual node or single node environments.

4 Redundant signaling subsystem architecture

Reference configurations

The Health Management system can support the following reference configurations:

- Single-node
- Dual-node
- Standalone

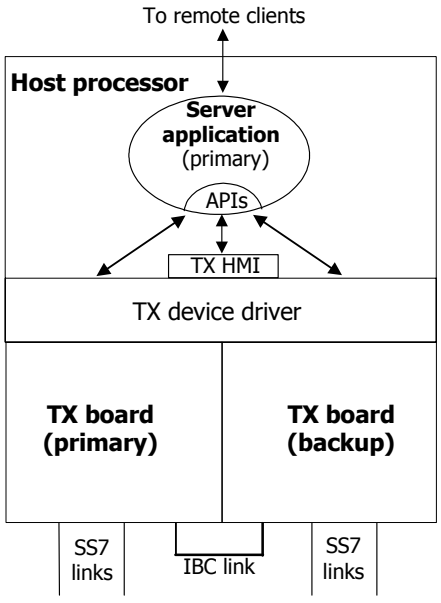
In the single-node and dual-node configurations, the signaling application is referred to as a signaling server, providing service to one or more signaling clients. The client-server model illustrated here is a common architecture for distributed call-processing applications, but others are possible. The choice of application model is up to the system designer.

This topic describes the Health Management system reference configurations. For SIGTRAN installations, replace the SS7 links with Ethernet links in the configuration illustrations.

For more information about setting up redundant configurations, refer to *Board installation and cabling* on page 73.

Single-node configuration

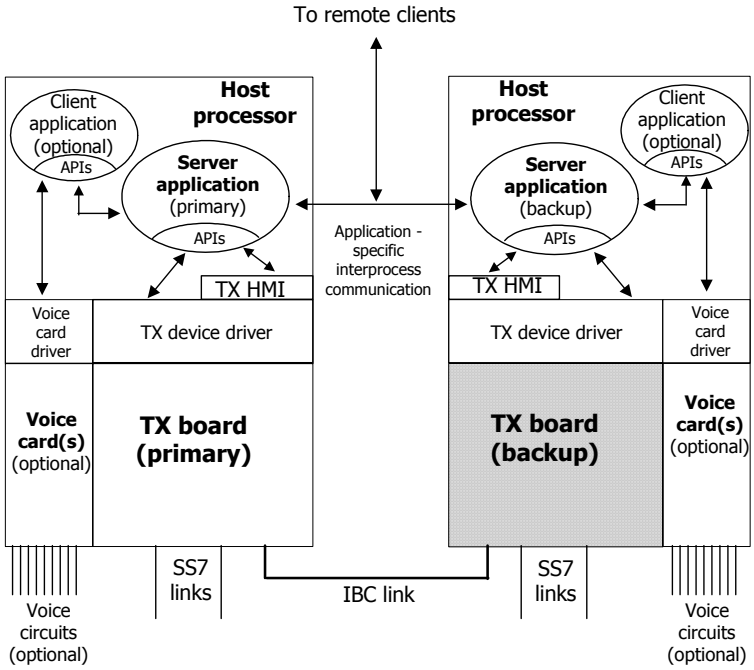
A single-node configuration uses two TX boards in a single-node (chassis) for board level redundancy. In this configuration, a single application monitors and controls the primary and backup boards and performs all other application functions. This is the simplest migration path for an existing non-redundant application to a redundant signaling subsystem.



A single-node redundant signaling subsystem can survive both signaling link and board failures without a service outage. In addition, one board can be taken out of service at a time for upgrade or reconfiguration without impacting the service provided by the application.

Dual-node configuration

A dual-node configuration uses two chassis, each with a single TX board for signaling. The dual-node model assumes a signaling server application that, like the TX boards, operates in a primary and backup manner. The primary and backup server applications communicate call states through an application-specific inter-process communication (IPC) mechanism.



A dual-node configuration has the reliability attributes of the single-node configuration but can also survive a failure or planned outage (for upgrade or reconfiguration) of an entire node without a service outage. The cost of this added reliability is in the increased complexity of the server applications. In a dual-node configuration, monitoring and control of the boards must be shared between applications on each node. If active calls or transactions are to be maintained across an outage, state information must also be exchanged between nodes.

Standalone configuration

A standalone configuration consists of a single non-redundant signaling board in a single node. In this configuration, the Health Management service monitors the board for failures and takes corrective action, such as reloading the failed board or notifying maintenance personnel.

A standalone configuration does not have the availability properties of a redundant configuration, but the Health Management service can still be a valuable tool for quickly detecting failures and minimizing the duration of the service outage that results.

Software architecture

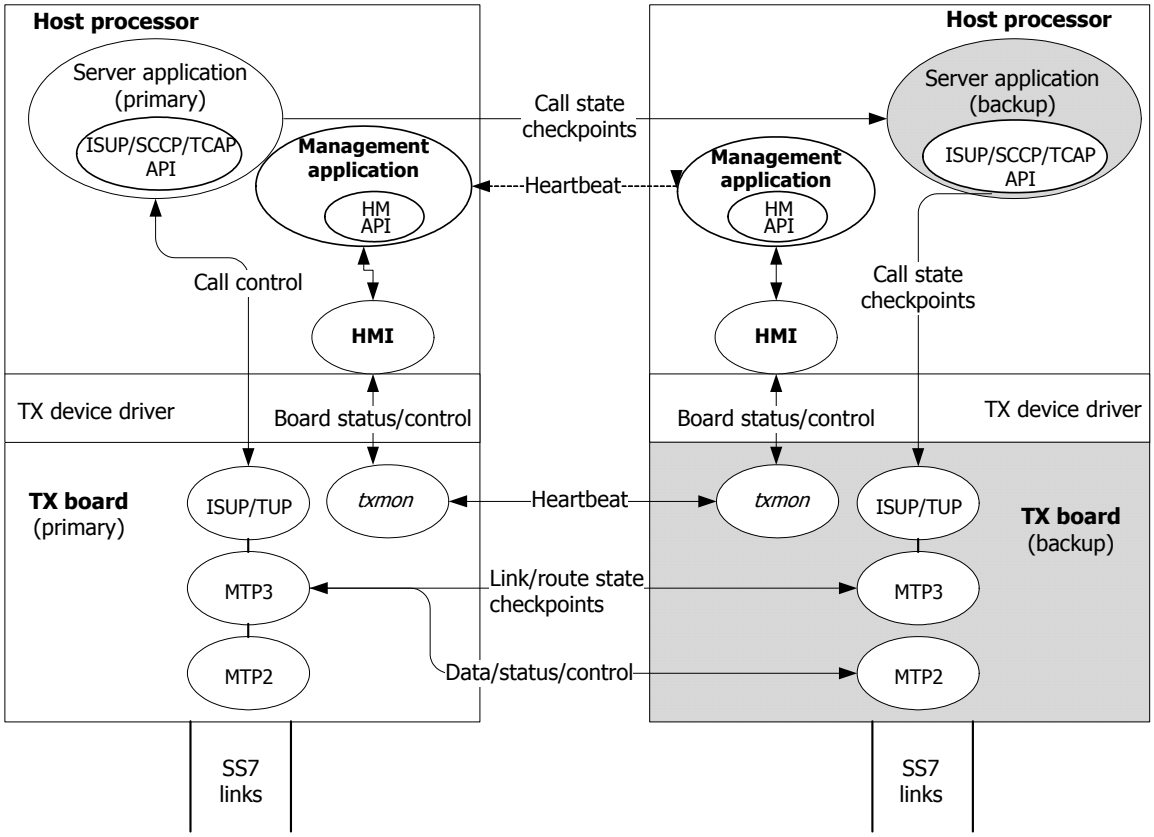
This topic shows the functional components and information flows for the following SS7 configurations:

- TDM
- IP

Although the illustrations show a dual node system, the components for a single node system are similar. TCAP and SCCP are also similar to this call processing example.

Software architecture for a TDM configuration

The following illustration shows the functional components and information flows in the software architecture model for an SS7 TDM configuration:



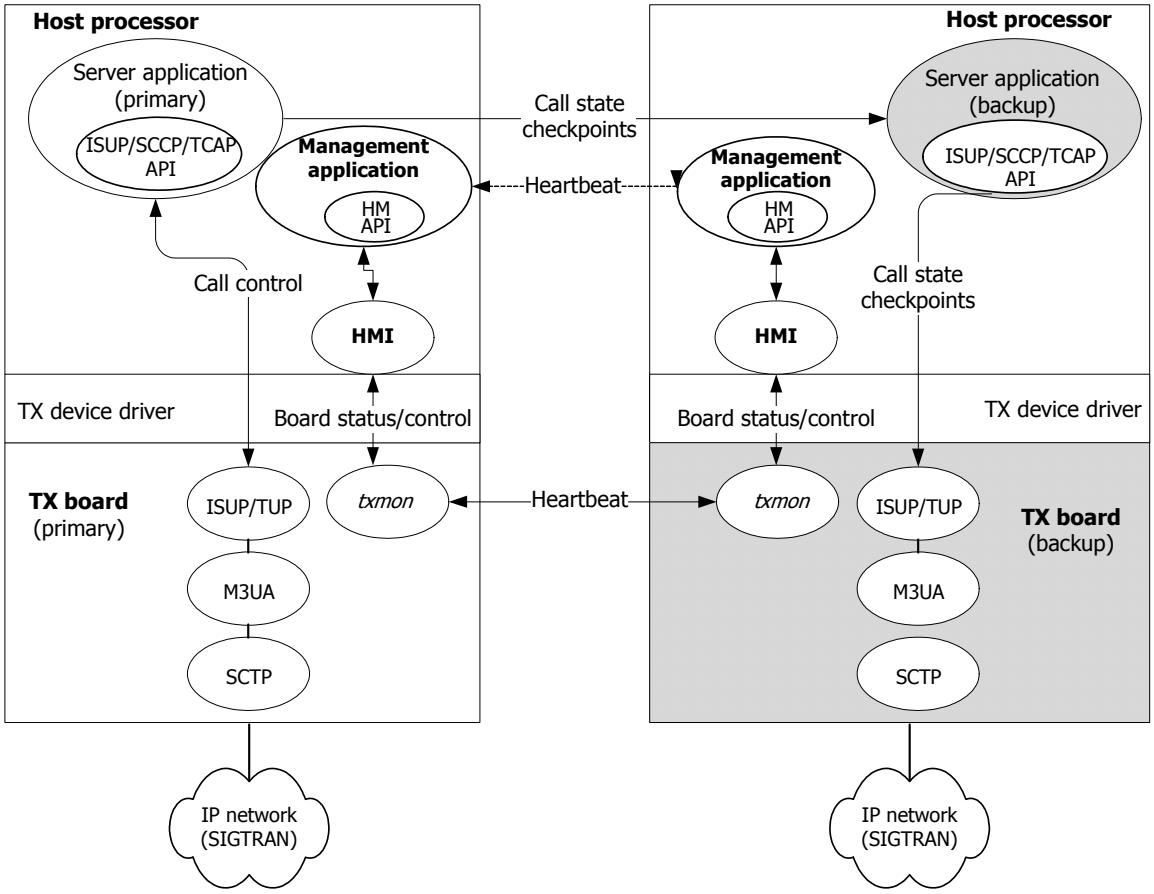
The following table describes each module and its information flows relating to high availability.

Module	Description
SS7 message transfer part (MTP)	SS7 MTP layers provide the physical signaling link termination, data link control, message routing, and network management functions for the signaling nodes. In a redundant configuration, active signaling links can be terminated on both boards. Both boards can be active up through MTP layer 2. All traffic received on either board is forwarded to the MTP 3 layer on the primary board for processing. All outgoing traffic is routed to the primary board by the application for delivery. The primary MTP 3 layer distributes outgoing traffic across all available links on both boards. Changes in the status of signaling links and routes are checkpointed by the primary MTP 3 layer to the backup MTP 3 layer so the backup is in the correct state in the event that it must become the primary. Refer to the *Dialogic® NaturalAccess™ MTP 2 Layer Developer's Reference Manual* and the *Dialogic® NaturalAccess™ MTP 3 Layer Developer's Reference Manual* for more information.
SS7 ISUP	The SS7 ISUP layer provides for the establishment, supervision, and clearing of all circuit-switched connections. In a redundant configuration, the primary board handles all live traffic. The backup ISUP layer remains in a state ready to assume control when needed. To preserve active calls in the case of a failure of the primary board, the call processing application can checkpoint updates of circuit states to the backup ISUP layer (through SS7 ISUP) as calls progress or as circuits become blocked and unblocked. SS7 ISUP includes a sample call processing application, *isupdemo*, which demonstrates the checkpointing of circuit states in various situations. For more information, refer to *ISUP demonstration program overview* on page 99. Refer to the *Dialogic® NaturalAccess™ ISUP Layer Developer's Reference Manual* for more information.
SS7 SCCP	SS7 SCCP provides services for routing non-circuit related traffic, including global title translations. In a redundant configuration, the primary node receives all live SCCP traffic. The backup SCCP takes over if there is a primary failure or switchover. The SCCP task checkpoints relevant routing information without any application involvement. Refer to the *Dialogic® NaturalAccess™ SCCP Layer Developer's Reference Manual* for more information.
SS7 TCAP	SS7 TCAP provides services for non-circuit related messaging, often destined for databases such as local number portability lookups. In a redundant configuration, the primary node receives all live TCAP traffic. The backup TCAP takes over if there is a primary failure or switchover. The TCAP task is configured to checkpoint some or all transactions. This is done automatically without any application responsibilities. SS7 TCAP includes a sample redundant application, *tcapdemo*, which demonstrates application behavior during transactions and switchovers. For more information on *tcapdemo*, refer to *TCAP demonstration program overview* on page 111. Refer to the *Dialogic® NaturalAccess™ TCAP Layer Developer's Reference Manual* for more information.

Module	Description
SS7 TUP	The SS7 TUP layer provides for the establishment, supervision, and clearing of all circuit-switched connections. In a redundant configuration, the primary board handles all live traffic. The backup TUP layer remains in a state ready to assume control when needed. To preserve active calls in the case of a failure of the primary board, the call processing application can checkpoint updates of circuit states to the backup TUP layer (through the standard TUP API) as calls progress or as circuits become blocked and unblocked. SS7 TUP includes a sample call processing application, *tupdemo*, which demonstrates the checkpointing of circuit states in various situations. For more information on *tupdemo*, refer to *TUP demonstration program overview* on page 117. Refer to the *Dialogic® NaturalAccess™ TUP Layer Developer's Reference Manual* for more information.
TX monitor (*txmon*)	The TX monitor (*txmon*) is a board-resident task that provides the health management functions on the TX boards. *txmon* functions include: • Monitoring the status of all registered SS7 tasks on the board and reporting any failures to the health management interface on the host. • Distributing and synchronizing the execution of board state requests (SET backup, SET primary, and so on) from the host. • Reporting changes in the status (connected, isolated) of the inter-board communications link to both the registered board-resident tasks and the health management interface. • Implementing the board side of the keep alive function that allows the health management interface to detect a board failure.
Health Management Interface (HMI) service	The Health Management Interface (HMI) is a host-based service (Windows) or daemon process (UNIX) that provides the actual execution of control functions requested by applications using the Health Management system. It supports multiple applications and distributes asynchronous board events to all registered applications. It also continuously monitors all configured boards to detect board failures.
Health Management service (HM API)	The Health Management service provides functions for applications to monitor and control the state of TX boards on their local machine. It provides functions to download a board, halt a board, retrieve the current state of a board, and set a board into primary or backup state. In addition, applications can register to receive asynchronous events indicating changes in the state of a board. These events include notifications that a board has failed, been downloaded or halted, set into the primary or backup states, or has become connected to or isolated from its mate board. The RMG demonstration program, included in the HMI distribution package, demonstrates the use of the Health Management service. The RMG demonstration program performs the role of the management application. For more information, refer to *RMG demonstration program overview* on page 93.

Software architecture for an IP configuration

The following illustration shows the functional components and information flows in the software architecture model for an SS7 IP configuration:



The following table describes each module and its information flows relating to high availability.

Module	Description
M3UA and SCTP	There are no checkpoint or data messages flowing between mate boards. Refer to the *Dialogic® NaturalAccess™ SIGTRAN Stack Developer's Reference Manual* for more information.
SS7 ISUP	The SS7 ISUP layer provides for the establishment, supervision, and clearing of all circuit-switched connections. In a redundant configuration, the primary board handles all live traffic. The backup ISUP layer remains in a state ready to assume control when needed. To preserve active calls in the case of a failure of the primary board, the call processing application can checkpoint updates of circuit states to the backup ISUP layer (through SS7 ISUP) as calls progress or as circuits become blocked and unblocked. SS7 ISUP includes a sample call processing application, *isupdemo*, which demonstrates the checkpointing of circuit states in various situations. For more information, refer to *ISUP demonstration program overview* on page 99. Refer to the *Dialogic® NaturalAccess™ ISUP Layer Developer's Reference Manual* for more information.

Module	Description
SS7 SCCP	SS7 SCCP provides services for routing non-circuit related traffic, including global title translations. In a redundant configuration, the primary node receives all live SCCP traffic. The backup SCCP takes over if there is a primary failure or switchover. The SCCP task checkpoints relevant routing information without any application involvement. Refer to the *Dialogic® NaturalAccess™ SCCP Layer Developer's Reference Manual* for more information.
SS7 TCAP	SS7 TCAP provides services for non-circuit related messaging, often destined for databases such as local number portability lookups. In a redundant configuration, the primary node receives all live TCAP traffic. The backup TCAP takes over if there is a primary failure or switchover. The TCAP task is configured to checkpoint some or all transactions. This is done automatically without any application responsibilities. SS7 TCAP includes a sample redundant application, *tcapdemo*, which demonstrates application behavior during transactions and switchovers. For more information on *tcapdemo*, refer to *TCAP demonstration program overview* on page 111. Refer to the *Dialogic® NaturalAccess™ TCAP Layer Developer's Reference Manual* for more information.
SS7 TUP	The SS7 TUP layer provides for the establishment, supervision, and clearing of all circuit-switched connections. In a redundant configuration, the primary board handles all live traffic. The backup TUP layer remains in a state ready to assume control when needed. To preserve active calls in the case of a failure of the primary board, the call processing application can checkpoint updates of circuit states to the backup TUP layer (through the standard TUP API) as calls progress or as circuits become blocked and unblocked. SS7 TUP includes a sample call processing application, *tupdemo*, which demonstrates the checkpointing of circuit states in various situations. For more information on *tupdemo*, refer to *TUP demonstration program overview* on page 117. Refer to the *Dialogic® NaturalAccess™ TUP Layer Developer's Reference Manual* for more information.
TX monitor (*txmon*)	The TX monitor (*txmon*) is a board-resident task that provides the health management functions on the TX boards. *txmon* functions include: • Monitoring the status of all registered SS7 tasks on the board and reporting any failures to the health management interface on the host. • Distributing and synchronizing the execution of board state requests (SET backup, SET primary, and so on) from the host. • Reporting changes in the status (connected, isolated) of the inter-board communications link to both the registered board-resident tasks and the health management interface. • Implementing the board side of the keep alive function that allows the health management interface to detect a board failure.
Health Management Interface (HMI) service	The Health Management Interface (HMI) is a host-based service (Windows) or daemon process (UNIX) that provides the actual execution of control functions requested by applications using the Health Management system. It supports multiple applications and distributes asynchronous board events to all registered applications. It also continuously monitors all configured boards to detect board failures.

Module	Description
Health Management service (HM API)	The Health Management service provides functions for applications to monitor and control the state of TX boards on their local machine. It provides functions to download a board, halt a board, retrieve the current state of a board, and set a board into primary or backup state. In addition, applications can register to receive asynchronous events indicating changes in the state of a board. These events include notifications that a board has failed, been downloaded or halted, set into the primary or backup states, or has become connected to or isolated from its mate board. The RMG demonstration program, included in the HMI distribution package, demonstrates the use of the Health Management service. The RMG demonstration program performs the role of the management application. For more information, refer to *RMG demonstration program overview* on page 93.

Board state model

For health management purposes, each board in a redundant pair is in one of several states, as described in the following table. Boards change state as a result of application commands issued through the Health Management service or other external events, such as hardware or software failures on the board.

State name	Description
Starting	Initial state of each board immediately after download, waiting for a command from the application to be primary or backup.
Primary	Board is active and is the primary member of a redundant board pair.
Backup	Board is active and is the backup member of a redundant board pair.
Shutdown	Reserved for future use.
Failed	Board is not operational due to a hardware or software failure or has been halted by an application. The application can attempt to reload the board.
Standalone	Board is either not equipped or not licensed for redundant operation and is running as a standalone signaling board.
Stopped	Board has been extracted.

Initialization

The signaling subsystem initialization phase involves:

- Downloading and configuring the board
- Setting it into the appropriate state (either primary or backup)
- Binding the applications and SS7 layers together

Downloading and configuring the board

Call **hmiLoadBoard** to initiate board download. Once a board download is initiated (including configuration), a HMI_EVN_LOADING event is sent to all applications registered with the HMI, including the application that initiated the download. Once the download is complete and the board is ready for operation, the application receives a HMI_EVN_STARTING event (or HMI_EVN_STANDALONE event, if the board is not operating in a redundant configuration).

The HMI cannot detect certain types of board download failures. Applications must time for the HMI_EVN_STARTING (or HMI_EVN_STANDALONE) event to detect a failed download. The duration of the timer could be anywhere from five seconds (for a normal sized configuration) to ten seconds or longer for a large configuration.

Setting the board state

After download, each board is initially in the starting state, waiting for **hmiPrimary** or **hmiBackup**. In the starting state, protocol tasks can be configured and bind requests can be honored, but links are not enabled and data traffic is not accepted. During this time, *txmon* attempts to establish communication with its mate board.

The application determines which board should be primary and which should be backup. Once determined, the application issues a **hmiPrimary** [**hmiBackup**] command to each board, as appropriate, through the HMI. Once the command is accepted, an HMI_EVN_NOWPRIMARY [HMI_EVN_NOWBACKUP] event is sent to all applications registered with the HMI, including the application that initiated the request.

Binding the applications and SS7 layers together

During this period each application also binds to its service provider layer through the appropriate function call. For example, call processing applications bind to the ISUP or TUP layer; direct MTP 3 applications bind to the MTP 3 layer. On a single node signaling subsystem, the application typically binds to its service provider on the primary board and the backup board. On a dual node signaling subsystem, the primary application binds to the primary board and the backup application binds to the backup board.

Upon a successful bind, the service user (application) is notified of the board status (primary or backup) through a status indication event. The service user must wait for the now primary status indication event before starting data traffic. This event precedes any incoming data traffic delivered to the service user and signals that normal data transfer can begin in either direction.

Hot Swap support

The health management system supports Hot Swap for CompactPCI installations. When the ejector handle is lowered to indicate the board is to be extracted, the HMI issues an HMI_EVN_EXTRACT event to all applications registered for receipt of asynchronous events.

A redundancy manager application receiving this event prepares any applications associated with the board for board removal. The controlling application calls **hmiStop**, which causes HMI to send an HMI_EVN_STOP event to all applications registered for receipt of asynchronous events.

Upon receipt of this event, all applications close their communications channels to the board and the Hot Swap LED lights, indicating that the board is ready for extraction.

When a board is inserted into the chassis, the HMI issues an HMI_EVN_INSERT event to all applications registered for receipt of asynchronous events. When the redundancy manager application receives this event, it initiates a load of the board.

Configuration and management

This topic explains how the hardware and software is configured and managed. It includes information about:

- Configuration utilities and functions
- Control, status, and statistics
- Alarms

Configuration utilities and functions

In general, each signaling board is loaded and configured independently. Each configuration utility and function call sends configuration packets to the target board. Configuration requests are not explicitly exchanged between boards. Each node in a multiple node signaling subsystem must have its own copy of each SS7 configuration file or database or be able to access a common file or database through networking. Similarly, each dynamic configuration function call must be executed on both boards.

The configurations downloaded to the SS7 layers are identical on both boards in a pair. To support configuration changes without a service outage, it is sometimes necessary to download a backup board with a new configuration, make it the primary board, and then reload the other board with the new configuration. In this case, the configurations on the two boards are out of synchronization for some time period. This is allowed during the checkpointing of state information between boards.

Control, status, and statistics

Control, status, and statistics requests are applied individually to each board in a mated pair. Control requests (enable or disable signaling links, block or unblock voice circuits, and so on) can only be issued to the primary board. Control requests issued to the backup board are rejected with an invalid state indication.

Status type requests can be issued to either the primary or the backup board. The results returned by a board reflect the status of the entity as currently viewed by that board. Thus, status requests issued to the backup board can be used to determine if an event, such as a call being answered, was checkpointed correctly to the backup.

Statistics requests can also be issued to either the primary or backup board. The statistics returned reflect events that occurred on that board only; no attempt is made to collate statistics between the primary and backup boards.

Alarms

Each board generates its own alarms. In a dual node signaling subsystem, the MTP 2 alarms associated with a particular link appear on the node that the link terminates on, not necessarily the active node. MTP 3, SCTP, M3UA, SCCP, TCAP, TUP, and ISUP alarms relating to operational events typically appear only on the active node.

The *txmon* task on each board generates alarms relating to the state of that board:

- Transitions between primary and backup mode
- Failures of tasks on the board
- Changes in the status of the inter-board link

5 Failure detection and recovery

Signaling link failures

Signaling link failures are handled within the MTP layers. Applications are not notified of signaling link failures unless the failure leaves a concerned destination unreachable (in which case it receives a pause event for the concerned destination) or the application/user part registered for link status events.

Signaling link recovery is automatic and transparent to MTP user parts and applications unless it results in a previously unreachable destination becoming reachable or the application/user part has explicitly registered for link status events.

In SIGTRAN configurations, loss of all associations triggers an HMI_EVN_NETWORK_DOWN event. A redundancy manager program, such as the RMG demonstration program, can use this event to switch over to the backup associations, allowing traffic to quickly resume.

Signaling board failures

A signaling board failure is detected by the HMI on the local signaling node. A failure can be a software failure on the board detected by the *txmon* process and reported to the HMI service, or a hardware failure such that the HMI service loses communication with the board. Both are reported to registered applications as board failures so that recovery can take place. Two different recovery scenarios are distinguished: failure of the primary board and failure of the backup board.

Primary board failure

When the application receives a HMI event indicating a failure of the primary board, it typically initiates a switchover to the backup signaling board by calling **hmiPrimary**. Call processing applications (or applications using other SS7 signaling services) then wait for the now primary status indication from its service provider before resuming data traffic.

Once the switchover is initiated, an application reloads the failed board with **hmiLoadBoard**. When the download is complete (HMI_EVN_STARTING event is received), the application sets the reloaded board into the backup state with **hmiBackup**. At that point, any SS7 service applications must rebind to their service providers. Any failed signaling links terminated on the reloaded board are automatically activated by the primary MTP 3. SIGTRAN associations are automatically re-established by the reloaded board if M3UA is configured as an ASP or IPSP client.

After the reload and rebind, the TUP, TCAP, and SCCP tasks automatically resynchronize with the primary TX board. The backup is then ready to take over operation. The backup ISUP layer considers all circuits to be idle. The application must re-synchronize the backup by checkpointing all non-idle circuit states through the ISUP service. The recovered board is then ready to take over the role of primary if needed.

If the board fails to reload cleanly (the HMI_EVN_STARTING event is not received within a reasonable time period), as might be the case with a true hardware failure, use **hmiHaltBoard** to stop the board. Manual intervention is required to recover the failed board.

Backup board failure

Failure of a backup board is detected and reported in the same fashion as the primary board. The board is typically reloaded (if possible) and set into the backup state. Applications must rebind with their service providers. Any failed signaling links terminated on the reloaded board are automatically activated by the primary MTP 3. SIGTRAN associations are automatically re-established by the reloaded board if M3UA is configured as an ASP or IPSP client.

After a backup with TUP, TCAP and/or SCCP is brought back into service, the application is ready to take over since TUP, TCAP, and SCCP automatically re-synchronize with the primary TX board.

Signaling node failures

Detection of signaling node failures in a dual-node configuration is application specific. No monitoring of the host or application status is done by the signaling subsystem. Recovery scenarios are similar to the failed board recovery scenarios.

Primary signaling node failure

When a primary signaling node fails, it is up to an application on the backup node to detect the failure and set the backup board into primary operation with **hmiPrimary**. During the primary node outage, messages arriving on signaling links terminated on the backup node are queued if possible, waiting for the switchover. If the traffic load is too heavy or the failure detection and recovery takes too long, the links can be placed in a local processor outage state and the queued messages may be lost.

After the failed signaling node is restored, the signaling board in the failed node is reloaded and placed into backup state. The sequence is the same as the recovery of a failed board except that additional synchronization is required between the applications on the primary and backup nodes to convey changes in circuit status that occurred while the failed node was unavailable. This synchronization is application specific.

Backup signaling node failure

Recovery of a failed backup signaling node is similar to the recovery of a failed backup board. No disruption of signaling traffic is expected in this case. If there is a total failure (the primary board detects the failure of the backup board), signaling links terminating on the failed board are declared failed until the backup board is restored. Blocking or resetting voice circuits that were terminated on the failed node is up to the application.

If only the backup host processor fails, the signaling links that terminate on the backup board remain operational until the backup node is rebooted.

The application is responsible for the synchronization of circuit states that changed while the backup node was out of service.

If the host fails in a signaling node but the TX board continues working, the TX board could become stranded. While this is acceptable when the backup node fails, some

action is necessary when the primary host fails. The primary TX board automatically becomes the backup if it sees its mate board become primary, resulting in no commands received from the Health Management service for approximately one half of a second.

Signaling board isolation

Signaling board isolation occurs when the inter-board link fails. In this case, neither board can communicate with the other and cannot distinguish this case from a failure of its mate board.

During isolation, the primary board keeps running but at (potentially) reduced capacity since the signaling links on the backup board cannot be accessed. Normal checkpointing of ISUP circuit states to the backup board from the host can still take place.

MTP configurations

When isolation is detected on the backup board, the active signaling links are put into an isolated state, queuing inbound packets but still delivering any queued outbound packets, and a short isolation timer is set. If the isolation ceases before the timer expires, normal traffic is resumed starting with the queued packets. If the isolation timer expires before the isolation condition is corrected, the isolated links are placed into the local processor outage (LPO) state and the queued inbound packets are discarded.

Switching the backup board into primary mode (as would be the case if the primary board failed) clears the isolated/LPO condition on those links and resumes normal traffic flow.

After the failed inter-board link is restored, the active MTP 3 layer clears the LPO condition on the isolated links to restore normal traffic and checkpoints any route or link states that may have changed during the isolation.

SIGTRAN configurations

There is no isolation timer or processor outage-like state associated with SIGTRAN. No action is taken upon an isolation event other than to log a message and change the isolatedState status field.

Planned switchovers

It may be necessary to remove a primary board from service to upgrade the software or hardware on the signaling node or the board itself. The recommended procedure is to manually switch the backup board into primary mode before shutting down the (now backup) board or node, as described here.

Once the applications have agreed that a switchover is necessary, the primary board is set into backup mode with **hmiBackup**. **hmiBackup** sets all signaling links into a flow-controlled state, resulting in all inbound packets being queued. Each layer (starting from MTP 3 or SIGTRAN on up) then sends a status indication (NOW BACKUP) to each of its service users.

Due to queuing between layers and within the device driver, the application can still receive some incoming signaling traffic between the issuing of the **hmiBackup** request and receipt of the NOW BACKUP status indication.

For ISUP messages, the following procedure is recommended:

- Connect Confirm (answer) and Release Confirm: Accept and checkpoint new circuit state to mate application/ISUP layer.
- All others: Discard and allow far end to timeout and retry if desired.

For TCAP, TUP, or SCCP messages, the following procedure is recommended:

- For TUP Connect Confirm (answer) and Release Confirm: Accept and checkpoint new circuit state to mate application.
- The application must ignore the event and wait for the other end to retry or (especially if the event is a checkpointed TCAP transaction or an SCCP connectionless) reply on the now primary board.

During this period, the application must not generate any new outbound signaling traffic.

Once the now backup status indication is received (indicating the end of any in-progress signaling traffic) the mate board is set to primary mode. This restarts the flow of signaling traffic to/from the mate board/node, including any messages queued within layer 2 during the switchover.

Note: Packets can be lost during a switchover. Heavy traffic during a switchover can result in either or both boards becoming congested due to the queuing of incoming packets. Planned switchovers are not recommended during periods of heavy load. Schedule maintenance during off-peak periods whenever possible.

6 Function reference

Function summary

Use the following functions with your application:

Function	Description
hmiBackup	Requests the board to switch to the backup signaling state.
hmiHaltBoard	Requests the local HMI to halt the local board.
hmiLoadBoard	Requests the local HMI to load the local board.
hmiPrimary	Requests the board to switch to the primary signaling state.
hmiReset	Requests the local HMI to read the configuration file again.
hmiShutdown	Requests the local HMI to shutdown gracefully, but without initiating a go backup message if active.
hmiStandalone	Requests the board to switch to the standalone signaling board.
hmiStart	Resumes communications to the TX board.
hmiStatusReq	Retrieves the current HMI status and statistics.
hmiStop	Disables communications to the TX board and sends an HMI_EVN_STOP event to all applications that are registered to receive unsolicited status events.

Using the function reference

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

Prototype	The prototype is followed by a listing of the function arguments. Data types include: • DWORD (8-bit unsigned) • S16 (16-bit signed) • U32 (32-bit unsigned) • Bool (8-bit unsigned) If a function argument is a data structure, the complete data structure is defined. **Note:** Not all parameters are applicable to both ANSI and ITU-T (CCITT) networks.
Return values	The return value for a function is either SUCCESS or an error code. For asynchronous functions, a return value of SUCCESS (zero) indicates the function was initiated; subsequent events indicate the status of the operation.

hmiBackup

Requests the board to switch to the backup signaling state. This request can only be issued to a board in the primary or starting state.

Prototype

DWORD **hmiBackup** (CTAHD ***ctahd***)

Argument	Description
ctahd	Natural Access handle from a previous successful call to **ctaOpenServices**.

Return values

Return value	Description
SUCCESS	
HMI_ERR_BADCMD	Request failed. An event handle was passed to the Health Management service command.
HMI_ERR_CLOSED	Request failed. Connection closed by HMI.
HMI_ERR_INTERROR	Request failed. Unknown I/O error occurred.
HMI_ERR_INVHANDLE	Handle is not valid.
HMI_ERR_NOMEM	No memory available.
HMI_ERR_REFUSED	Request failed because the board is in a halted or standalone state.
HMI_ERR_TIMEOUT	Request failed. No response from HMI.

hmiHaltBoard

Requests the local HMI to halt the local board.

Prototype

DWORD **hmiHaltBoard** (CTAHD ***ctahd***)

Argument	Description
ctahd	Natural Access handle from a previous successful call to **ctaOpenServices**.

Return values

Return value	Description
SUCCESS	
HMI_ERR_BADCMD	Request failed. An event handle was passed to the Health Management service command.
HMI_ERR_CLOSED	Request failed. Connection closed by HMI.
HMI_ERR_INTERROR	Request failed. Unknown I/O error occurred.
HMI_ERR_INVHANDLE	Handle is not valid.
HMI_ERR_NOMEM	No memory available.
HMI_ERR_TIMEOUT	Request failed. No response from HMI.

hmiLoadBoard

Requests the local HMI to load the local board. For more information, refer to *HMI configuration* on page 80.

Prototype

DWORD **hmiLoadBoard** (CTAHD ***ctahd***)

Argument	Description
ctahd	Natural Access handle from a previous successful call to **ctaOpenServices**.

Return values

Return value	Description
SUCCESS	
HMI_ERR_BADCMD	Request failed. An event handle was passed to the Health Management service command.
HMI_ERR_CLOSED	Request failed. Connection closed by HMI.
HMI_ERR_INTERROR	Request failed. Unknown I/O error occurred.
HMI_ERR_INVHANDLE	Handle is not valid.
HMI_ERR_NOMEM	No memory available.
HMI_ERR_TIMEOUT	Request failed. No response from HMI.

hmiPrimary

Requests the board to switch to the primary signaling state. This request can only be issued to a board in the backup or starting state.

Prototype

DWORD **hmiPrimary** (CTAHD ***ctahd***)

Argument	Description
ctahd	Natural Access handle from a previous successful call to **ctaOpenServices**.

Return values

Return value	Description
SUCCESS	
HMI_ERR_BADCMD	Request failed. An event handle was passed to the Health Management service command.
HMI_ERR_CLOSED	Request failed. Connection closed by HMI.
HMI_ERR_INTERROR	Request failed. Unknown I/O error occurred.
HMI_ERR_INVHANDLE	Handle is not valid.
HMI_ERR_NOMEM	No memory available.
HMI_ERR_REFUSED	Request failed because the board is in a halted or standalone state.
HMI_ERR_TIMEOUT	Request failed. No response from HMI.

hmiReset

Requests the local HMI to read the configuration file again.

Prototype

DWORD **hmiReset** (CTAHD ***ctahd***)

Argument	Description
ctahd	Natural Access handle from a previous successful call to **ctaOpenServices**.

Return values

Return value	Description
SUCCESS	
HMI_ERR_BADCMD	Request failed. An event handle was passed to the Health Management service command.
HMI_ERR_CLOSED	Request failed. Connection closed by HMI.
HMI_ERR_INTERROR	Request failed. Unknown I/O error occurred.
HMI_ERR_INVHANDLE	Handle is not valid.
HMI_ERR_NOMEM	No memory available.
HMI_ERR_TIMEOUT	Request failed. No response from HMI.

Details

A reset causes the HMI to re-read its configuration file. You can reconfigure by editing the HMI configuration file and issuing a reset request. However, the ports assigned to the HMI are not reconfigured on a reset. The HMI must be stopped and restarted to change ports.

All existing application connections are preserved during a reset.

hmiShutdown

Requests the local HMI to shutdown gracefully, but without initiating a backup request if active.

Prototype

DWORD **hmiShutdown** (CTAHD ***ctahd***)

Argument	Description
ctahd	Natural Access handle from a previous successful call to **ctaOpenServices**.

Return values

Return value	Description
SUCCESS	
HMI_ERR_BADCMD	Request failed. An event handle was passed to the Health Management service command.
HMI_ERR_CLOSED	Request failed. Connection closed by HMI.
HMI_ERR_INTERROR	Request failed. Unknown I/O error occurred.
HMI_ERR_INVHANDLE	Handle is not valid.
HMI_ERR_NOMEM	No memory available.
HMI_ERR_TIMEOUT	Request failed. No response from HMI.

Details

hmiShutdown causes the HMI service to shutdown immediately. If the board that is shut down is the primary board, no HMI service backup request is performed, since shutting down does not change the status of the board. If the user wants to immediately switchover before the primary board shuts down, the HMI service backup request must be issued before the shutdown request and an HMI primary request must be issued for the other board.

All applications registered for unsolicited status events are notified of the pending shutdown with a HMI_EVN_SERVICE_DOWN event.

hmiStandalone

Requests the board to switch to the standalone signaling board. This request can only be issued to a board in the starting or standalone state.

Prototype

DWORD **hmiStandalone** (CTAHD ***ctahd***)

Argument	Description
ctahd	Natural Access handle from a previous successful call to **ctaOpenServices**.

Return values

Return value	Description
SUCCESS	
HMI_ERR_BADCMD	Request failed. An event handle was passed to the Health Management service command.
HMI_ERR_CLOSED	Request failed. Connection closed by HMI.
HMI_ERR_INTERROR	Request failed. Unknown I/O error occurred.
HMI_ERR_INVHANDLE	Handle is not valid.
HMI_ERR_NOMEM	No memory available.
HMI_ERR_REFUSED	Request failed because the board is in a halted, primary, or backup state.
HMI_ERR_TIMEOUT	Request failed. No response from HMI.

hmiStart

Resumes communications to the TX board. This function cancels the affects of a previous call to **hmiStart**.

Prototype

DWORD **hmiStart** (CTAHD ***ctahd***)

Argument	Description
ctahd	Natural Access handle returned by **ctaOpenServices**.

Return values

Return value	Description
SUCCESS	
HMI_ERR_BADCMD	Request failed. An event handle was passed to the Health Management service command.
HMI_ERR_CLOSED	Request failed. Connection closed by HMI.
HMI_ERR_INTERROR	Request failed. Unknown I/O error occurred.
HMI_ERR_INVHANDLE	Handle is not valid.
HMI_ERR_NOMEM	No memory available.
HMI_ERR_TIMEOUT	Request failed. No response from HMI.

hmiStatusReq

Retrieves the current HMI status and statistics.

Prototype

DWORD **hmiStatusReq** (CTAHD ***ctahd***, HmStatsData ****pHmStats***, U8 ***reset***)

Argument	Description
ctahd	Natural Access handle from a previous successful call to **ctaOpenServices**.

pHmStats	Pointer to the statistics structure that is local to application memory:
	`typedef struct hmLnkStats`
	`{`
	`  U32 txHB;  /* number of link heartbeat messages transmitted */`
	`  U32 rxHB;  /* number of link heartbeat messages received */`
	`  U8  linkState; /* state of the IBC link, see hmidef.h */`
	`  U8  fill[3];`
	`} HmLnkStats;`
	
	`typedef struct hmTskStats`
	`{`
	`  U32 txLnkStInd;  /* number of link state change ind. transmitted */`
	`  U32 txRunStInd;  /* number of task state change ind. transmitted */`
	`  U32 rxHB;  /* number of link heartbeat mesg recvd from task */`
	`  U8  runState;  /* run state of task,incl HMRS_FAILED, see hmidef.h */`
	`  U8  rxSeq;  /* RX sequence number */`
	`  U8  txSeq;  /* TX sequence number */`
	`  U8  remRunState; /* run state of remote task of same name */`
	`  U8  name[8];  /* task name */`
	`} HmTskStats;`
	
	`typedef struct hmStatsData`
	`{`
	`  HmLnkStats linkStats;  /* statistics for the IBC link */`
	`  U8  numTsks;  /* number of tasks with statistics reported */`
	`  U8  netState;  /* network connectivity state */`
	`  U8  fill[2];`
	`  HmTskStats taskStats[HM_MAXTASKS]; /* task statistics */`
	`} HmStatsData;`

reset	If set to zero, do not reset counters. If non-zero, reset all counters.

The netState field in the HmStatsData structure applies to SIGTRAN configurations only. Valid values for netState are:

- HMNS_CONNECTED: Board network connectivity state is UP.
- HMNS_ISOLATED: Board network connectivity state is DOWN.
- HMNS_UNKNOWN: Board has not reported a network connectivity state.

Return values

Return value	Description
SUCCESS	
HMI_ERR_BADCMD	Request failed. An event handle was passed to the Health Management service command.
HMI_ERR_CLOSED	Request failed. Connection closed by HMI.
HMI_ERR_INTERROR	Request failed. Unknown I/O error occurred.
HMI_ERR_INVHANDLE	Handle is not valid.
HMI_ERR_NOMEM	No memory available.
HMI_ERR_REFUSED	Request failed because the board is in a halted state.
HMI_ERR_TIMEOUT	Request failed. No response from HMI.

hmiStop

Disables communications to the TX board and sends an HMI_EVN_STOP event to all applications that are registered to receive unsolicited status events. Call this function upon receipt of an HMI_EVN_EXTRACT event, indicating that a board is to be removed.

Prototype

DWORD **hmiStop** (CTAHD ***ctahd***)

Argument	Description
ctahd	Natural Access handle returned by **ctaOpenServices**.

Return values

Return value	Description
SUCCESS	
HMI_ERR_BADCMD	Request failed. An event handle was passed to the Health Management service command.
HMI_ERR_CLOSED	Request failed. Connection closed by HMI.
HMI_ERR_INTERROR	Request failed. Unknown I/O error occurred.
HMI_ERR_INVHANDLE	Handle is not valid.
HMI_ERR_NOMEM	No memory available.
HMI_ERR_TIMEOUT	Request failed. No response from HMI.

7 Developing an ISUP or TUP redundant application

Checkpointing strategies for ISUP or TUP applications

To preserve stable calls across outages, the primary application needs to checkpoint circuit and hardware state information to the backup application. The backup application checkpoints the received circuit state information to the ISUP stack on the backup board. TUP does not require this update to be sent to the stack on the board.

This section discusses some of the issues and strategies for maintaining consistent and accurate circuit state information across primary and backup applications and ISUP or TUP stacks.

Checkpoint information

This topic describes the checkpointing process, including:

- Backup application
- Transient state
- Incremental checkpointing
- Batch checkpointing

Backup application

The backup application is responsible for checkpointing circuit state information to the ISUP implementation running on the backup board. This information consists of the call processing state and circuit blocking state. This is the minimum information to be checkpointed from the primary application to backup application.

Upon receipt of checkpoint information, the backup application must checkpoint the circuit state information (call processing and circuit blocking states) to the ISUP stack on the backup board. This is accomplished through a call to **ISUPStatusReq** with an ***eventType*** of CIRGRPSET.

Transient state

Optionally, the primary application can checkpoint the transient state of a circuit. A circuit is in a transient state during call setup and call release. Checkpointing the transient state of a circuit allows the backup application to identify the circuits that were in call setup or release at the time of changeover. This information is not checkpointed to the ISUP stack.

Note: After changeover, the backup (now primary) application resets all circuits in a transient state.

Incremental checkpointing

Incremental checkpointing refers to the transmission of state information for a single circuit concurrent with a change in state for the referenced circuit.

This is the recommended checkpoint method for the following reasons:

- Backup application always has up to date information concerning all circuits.
- Transient circuit checkpointing is reasonable due to the timely update of information.
- Minimum number of circuit resets upon changeover.
- Acceptable for all size configurations.

Batch checkpointing

Batch checkpointing refers to the mass transmission, from primary application to backup application, of state information for controlled circuits and hardware.

Periodic batch checkpointing refers to the periodic transmission of state information for all controlled circuits and hardware.

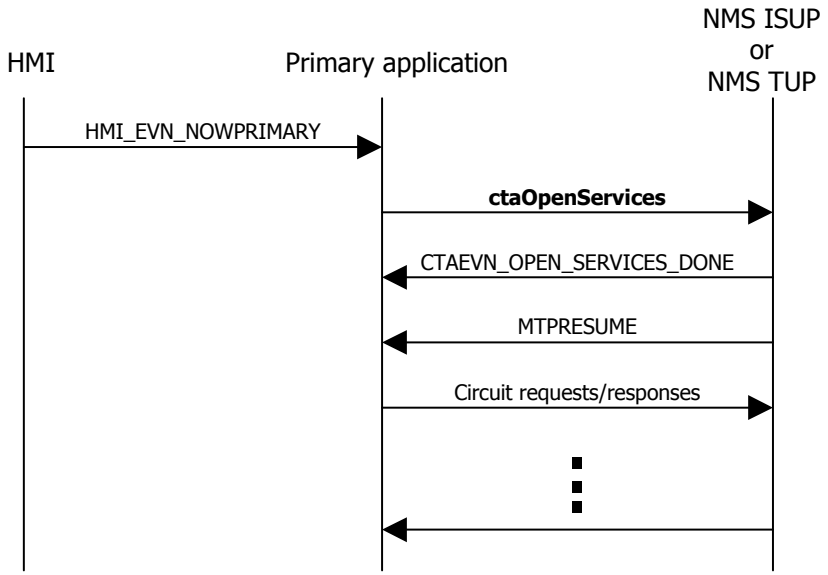
Periodic delta batch checkpointing refers to the periodic transmission of state information for controlled circuits and hardware that have changed state since the previous checkpoint. This is more efficient than normal periodic batch checkpointing.

The use of batch checkpointing is discouraged for several reasons:

- Large window of uncertainty; circuit activity between checkpoints is not visible to backup.
- Depending on the time between checkpoints, transient circuit checkpointing may not be effective.
- After changeover, transient circuits are unusable until protocol recovery or until all idle circuits are reset by the new primary.

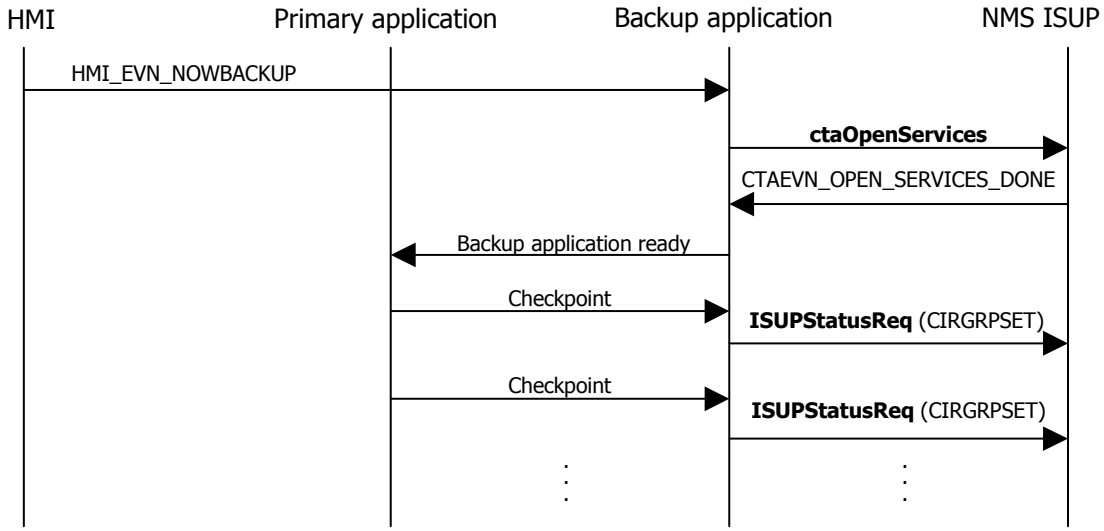
ISUP or TUP application startup

At startup, the backup and primary applications must open the ISUP or TUP service with a call to **ctaOpenServices**:



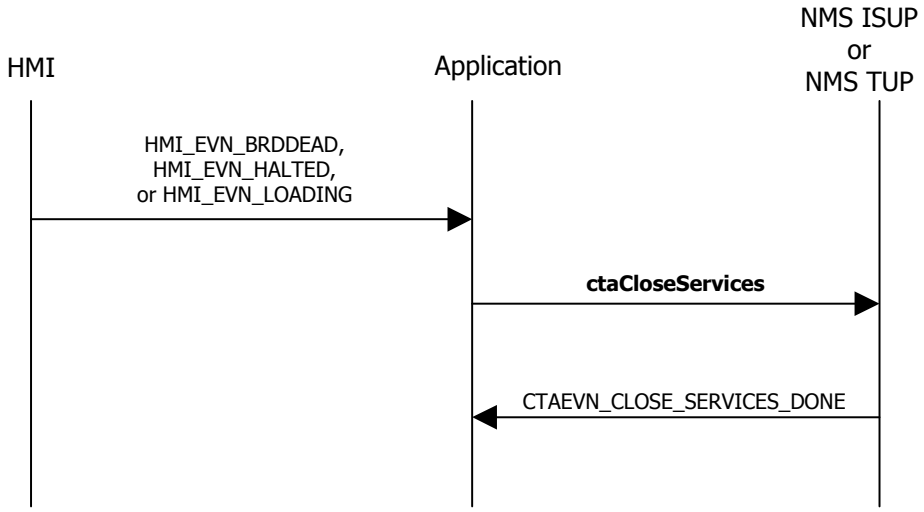
Note: Circuits default to a flow-controlled state. Until an MTPRESUME indication is received indicating that circuits are available, requests referencing these circuits return an error indication with cause 27: no route to destination.

Upon startup, the backup application receives a batch checkpoint from the primary application. As shown in the following illustration, the backup application checkpoints this information to the ISUP stack running on the backup board:



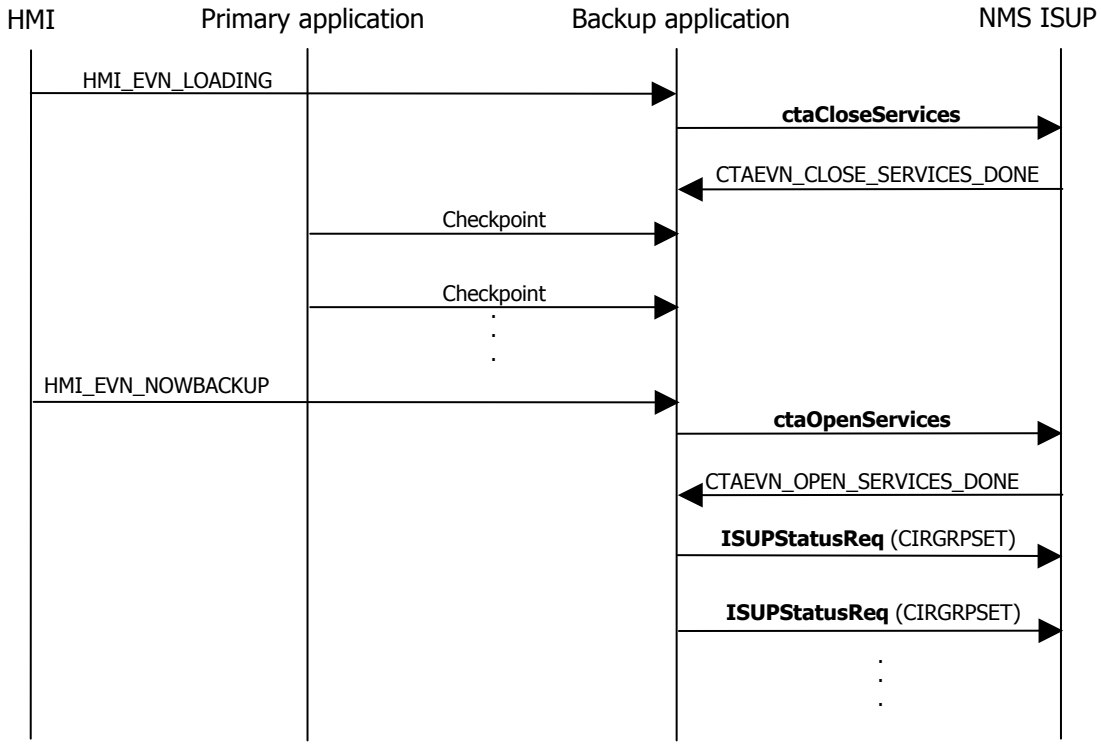
ISUP or TUP board failure, halt, or load

Upon notification of board failure, halt, or load, the applications close the ISUP or TUP service with a call to **ctaCloseServices**:



ISUP or TUP backup reload

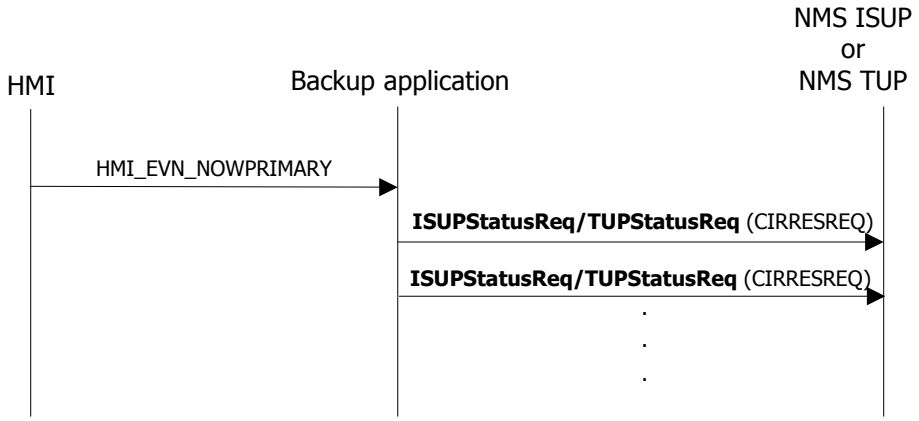
Upon notification that the backup board was reloaded and set to backup, the backup application reopens the ISUP or TUP service. In the case of ISUP, the backup application checkpoints the state of all circuits to the ISUP stack running on the backup board.



ISUP or TUP switchover

At switchover, the backup application (now the primary) resets all transient circuits if checkpointing of the transient state was implemented. If transient state was not maintained, the new primary application can either:

- Reset all idle circuits.
- Allow ISUP or TUP to recover transient circuits. This could take considerable time, depending upon what state the circuit was in at the time of switchover. Any attempt at using one of these circuits before the protocol has returned the circuit to idle state results in an error returned from ISUP or TUP.



Assume that the instance ID parameters (suInstId and spInstId) are set to 0 after a switchover.

8 Developing a TCAP redundant application

TCAP redundancy support

Two TX boards can be configured as a redundant pair. In this configuration, one board is designated as the primary board 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 redundancy, 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. This process 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.

A TCAP application can control transaction checkpointing by configuring default checkpoint behavior or by specifying checkpoint behavior for each transaction.

The TCAP application must also monitor several new indications that inform the application of the underlying board's state. These indications allow an application to correctly handle board failures, planned outages, and so on.

Refer to the *Dialogic® NaturalAccess™ TCAP Layer Developer's Reference Manual* for more information.

Handling TCAP traffic

In a redundant system, the TCAP application calls **ctaOpenServices** on both the primary and backup boards. TCAP traffic can only be sent and received on the primary board. By monitoring the run state indications from the TCAP service, the TCAP application can always determine which board to send TCAP traffic on.

Redundancy indications

Several indications support TCAP redundancy for TCAP applications. Some of these indications are generated by the TCAP task and are sent to TCAP applications. Others are generated by the Health Management service.

A TCAP application opens the TCAP service and the Health Management service.

TCAP task indications

Once a TCAP application opens the TCAP service, it receives a run state indication (primary, backup, or standalone) from the TCAP task. If the board state changes (from backup to primary), another run state indication is sent to the application from the TCAP task.

The backup TCAP task sends the backup ready indication only after it reconnects to the primary board or is reloaded. This indication signifies the backup task has a complete copy of the primary TCAP task's open transactions.

Health Management indications

The Health Management service provides indications that are useful for monitoring board status.

Ignore the following HM run state indications:

- Now primary
- Now backup
- Now standalone

A TCAP application waits for the run state indication from the TCAP task before modifying its behavior. However, the Health Management service run state indication can be useful as a signal to open the TCAP service after a board loads.

The application closes the TCAP service if any of the following Health Management service indications are received:

- Board dead
- Board halted
- Board loading
- Task dead
- Stop

The stop indication is only received when a CompactPCI board is removed from its chassis.

The following indications monitor the state of the primary TX board to the backup TX board private Ethernet link:

- Connected
- Isolated

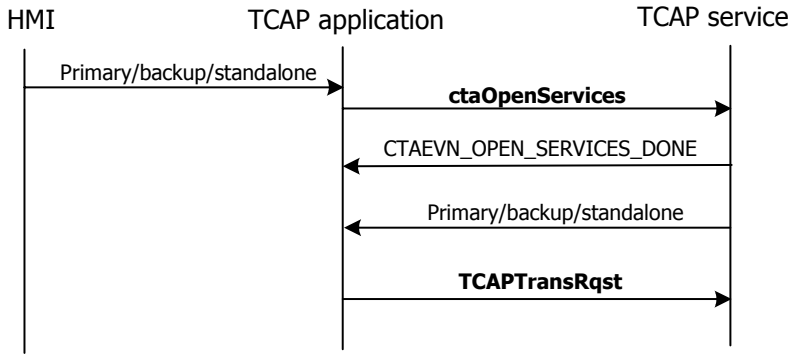
If an isolated indication is received, active transactions are not being checkpointed.

Note: The indications listed here are not a complete list of the available Health Management service indications, but only those applicable to TCAP applications.

TCAP board load

After loading a board and receiving a Health Management service run state indication (primary, backup, or standalone), a TCAP application can open the TCAP service.

Once the TCAP service is successfully opened, a run state indication from the TCAP task is received. At this point, TCAP traffic can begin on the primary (or standalone) board.



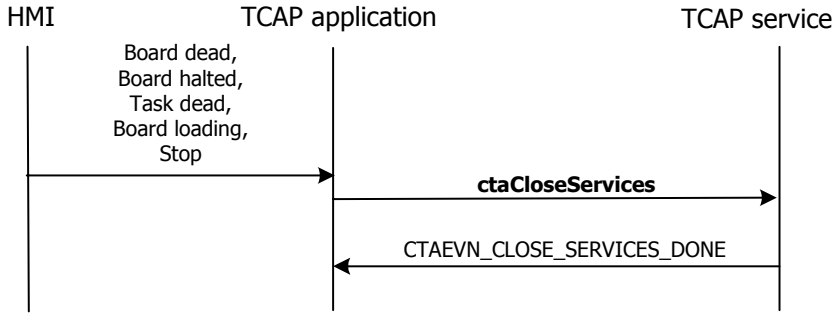
TCAP board failure

A TCAP application closes the TCAP service when any of the following indications are received:

- Board dead
- Board loading
- Board halted
- Task dead
- Stop

In these cases, the TCAP application closes the TCAP service and begins the process of reloading the board.

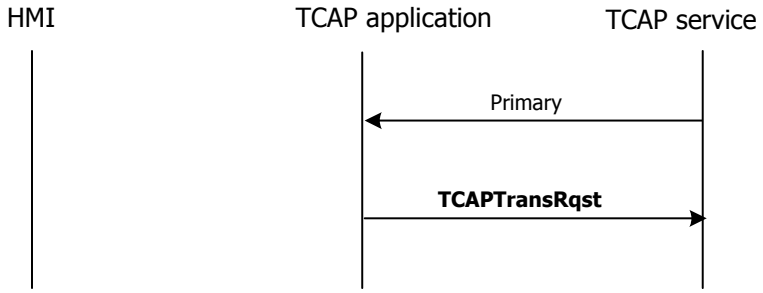
The TCAP application only closes the TCAP service once to a TX board. This can be performed when a failure occurs or when the board is reloaded.



TCAP switchover

The primary board and backup board can be switched in response to an application request. This is known as a switchover.

When the primary run state indication is received from the TCAP task, the TCAP application sends all traffic to the new primary board. In this case, ignore the Health Management service primary indication.

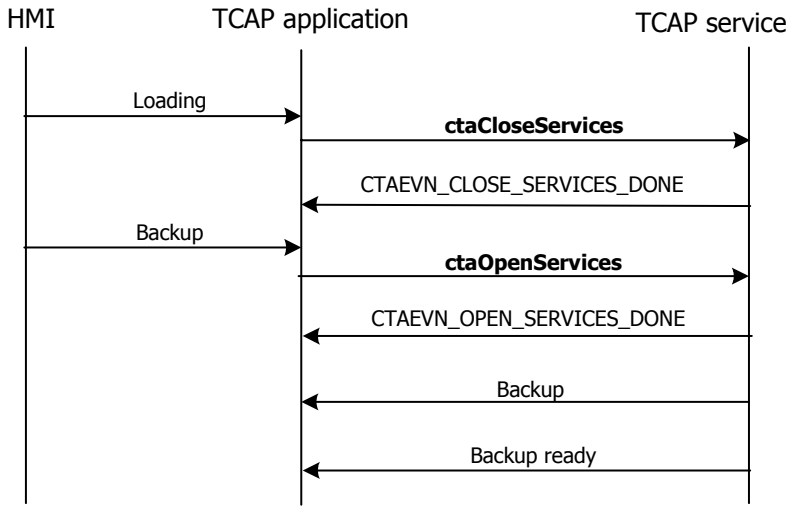


TCAP backup reload

When a failed board is reloaded, the TCAP application closes the TCAP service on that board if it was not already closed when the failure occurred.

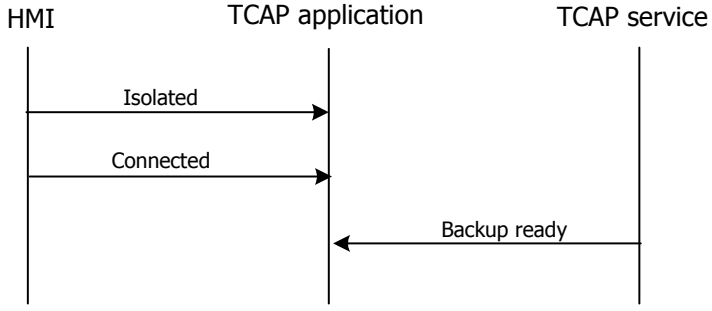
When the Health Management service backup indication is received, the TCAP application can reopen the TCAP service to the board. A backup run state indication is received from the TCAP service after the open service is completed.

When the backup board is reloaded, it no longer contains a valid set of open transactions. After the backup board is reloaded, it requests an update of the open transactions from the primary board. When this process is complete, a backup ready run state indication is sent to the TCAP application.



TCAP backup isolation

If the backup board becomes isolated from the primary board, it no longer contains a valid set of open transactions. After the backup board is reconnected, it requests an update of the open transactions from the primary board. When this process is complete, a backup ready run state indication is sent to the TCAP application.



9 Developing a SCCP redundant application

SCCP layer overview

The SCCP layer can operate in a two-board redundant primary or backup configuration in addition to the single-board standalone configuration. The boards can be in the same chassis or they can be in two chassis.

The objective of the redundant configuration is to maintain the SCCP service if any of the following scenarios occur:

- The primary board fails (hardware or software failure)
- The chassis containing the primary board fails (dual-node configuration)
- A planned outage for maintenance purposes

Sufficient state information regarding the primary SCCP layer must be reflected in the backup SCCP layer so that operation can continue without a service disruption. This state information includes the status of remote signaling points and subsystems, the status of local subsystems, and the state of any active class 2 or class 3 connections.

The SCCP layer automatically maintains the correct state of its backup by checkpointing state information whenever necessary, such as when the state of a remote signaling point or subsystem changes or when a new connection is established (confirmed). It also restores the current state to the backup whenever the backup is reloaded or recovers from isolation. These operations are transparent to the application.

Connectionless services

For both class 0 and class 1 connectionless services, best effort delivery service is maintained across switchovers. No state information is maintained between primary and backup SCCP layers other than the status of remote signaling points or subsystems.

It is possible for messages to be lost on a switchover for both classes of connectionless service. The application-level protocol running above SCCP is responsible for detecting and recovering lost messages.

For both classes of service, segmented messages in the process of being transmitted or received are lost or discarded on a switchover. If the remaining segments of a partially reassembled incoming message that was lost or discarded due to a switchover are received by the (new) primary, they are detected and discarded. If any of these segments has the return option set, the segment is returned to the sender in an XUDTS message with a return cause of Segmentation Failed for ITU or Error in Message Transport for ANSI.

Connection-oriented services

Confirmed connections (class 2 and class 3) are maintained across switchovers. The following table describes what happens to a connection during connection-oriented switchover:

Connection type	Description	Connection status
Transient connection	Connection is being established or cleared	Dropped on a switchover
Connections that look active to the far end but idle to the local side	A connection confirmation is returned but the checkpoint did not make it to backup, or a release was initiated but the far end did not receive it	Cleared through the SCCP protocol-level inactivity timers

The backup SCCP layer maintains the frozen status of each source local reference (SLR) assigned by the primary SCCP layer, so that the SLR is not re-used for the appropriate period of time after a switchover.

Message status

The following table describes what happens to messages during connection-oriented switchover:

Message is...	Description	The message...
In transit at the time of a switchover	Can be traveling in either direction	May be lost
Segmented and being reassembled at the time of a switchover	Is incoming	Will be dropped
Segmented for class 2 connections	The SCCP layer cannot detect that subsequent segments belong to a previous message.	Is delivered to the application as a new message. Detection of the incomplete message and any recovery is the responsibility of the application.
Segmented for class 3 connections	The SCCP layer can detect and discard any subsequent segments received.	

Connection information

Application inactivity control and connection auditing help synchronize connection information between primary and backup SCCP layers and applications.

Connection information mechanism	Description
Application inactivity control	Complements the SCCP protocol level inactivity control by allowing the SCCP layer to detect that an active connection at the stack is no longer active at the application level. Application inactivity control must be enabled by a configuration option.
Connection auditing	Allows the application to retrieve a list of all active connections maintained by the SCCP layer. This is not recommended to replace application-level checkpointing of connection information, but can be useful when a backup application has failed and been restarted or as a general-purpose robustness mechanism.

Refer to the *Dialogic® NaturalAccess™ SCCP Layer Developer's Reference Manual* for more information.

Redundant application models

The redundant application models are:

- Single-node
- Dual-node (distributed)

Single-node redundant application model

In a single-node application model, both boards reside in the same chassis and a single application instance opens and simultaneously binds to the same SCCP user SAPs on both boards.

At any given point in time one board is primary, the other is backup. The single-node application tracks which board is primary and directs all traffic toward the primary board.

Because a single application instance handles all traffic to and from both boards, no external mechanism is required to synchronize state information between application instances.

Dual-node redundant application model

In a dual-node application two instances of the same application exist, typically on separate chassis. Each instance of the application binds to a single board and, at any given point in time, one instance of the application is the primary and one is the backup (corresponding to the board states).

Only the primary application instance sends and receives network traffic; the backup monitors the state of the primary, waiting to become primary itself.

In this model, some external mechanism may be needed to synchronize the state of the primary and backup application instances - this is up to the application. For example, if connection-oriented services are employed, the primary application instance may need to checkpoint the connection state to the backup application instance so the backup can preserve active (confirmed) connections across a switchover.

Note: It is possible to implement a single-node system employing two application instances, one for each board. The considerations for this case are the same as for a dual-node architecture (although the implementation may be different).

In either application model, the application must register for the HMI service and the SCCP service to fully support redundant operation.

SCCP application considerations

There are application considerations for responding to redundancy-related events in SCCP service applications. In many cases, the considerations are the same for both single-node and dual-node applications.

Note: The message flow illustrations in these topics show separate application instances for each board. For a single-node application architecture, this can be thought of as separate contexts within a single application instance.

The application considerations are:

- SCCP redundant application startup
- SCCP normal operation
- SCCP switchovers
- SCCP board failure and reload

SCCP redundant application startup

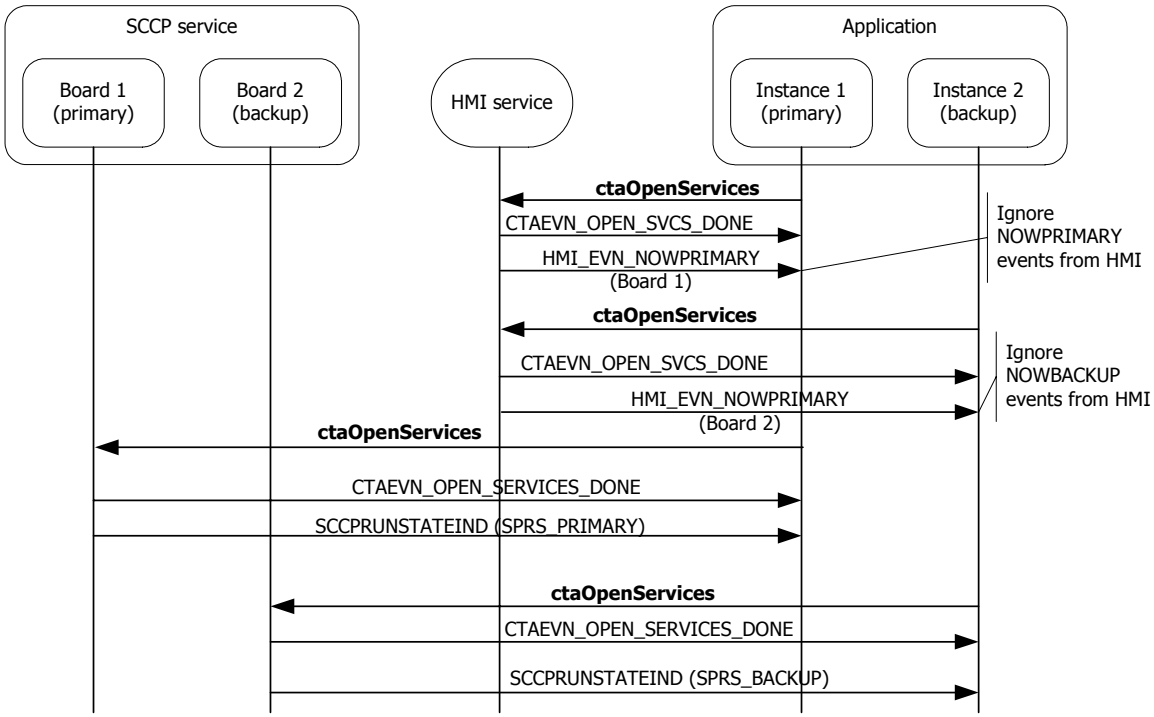
Each application instance performs the following operations when starting up an SS7 SCCP redundant application:

Step	Action
1	Creates one or more CTA queues on which to receive events. A single queue can be used for SCCP and HMI service events, or separate queues can be used, depending on the organization of the application (single-threaded versus multi-threaded).
2	Creates CTA contexts for the HMI service and for each instance of the SCCP service being opened. For a single-node application opening both boards in a redundant pair, a separate context must be created for each SCCP user SAP (SSCP subsystem) being opened on each board. These contexts can be assigned to the same or to separate queues.
3	Opens the appropriate service (SCCP or HMI) for each context and waits for the CTAEVN_OPEN_SERVICES_DONE event. If a single queue is used for multiple contexts, be prepared to process events from other contexts while waiting for the CTAEVN_OPEN_SERVICES_DONE event; failure to do so may result in an infinite loop.
4	Waits for the SCCPRUNSTATEIND event from the SCCP service before starting traffic. If the SCCP layer on the target board is already in the primary or backup state, a SCCPRUNSTATEIND event is immediately delivered to the application identifying the state (event type = SPRS_PRIMARY or SPRS_BACKUP). If still in the starting state (has not yet been designated primary or backup by the system manager application), a SCCPRUNSTATEIND event is not generated until the board is set to primary or backup state. If the board is operating in a standalone (non-redundant) configuration, the application immediately receives a SCCPRUNSTATEIND event with an event type of SPRS_STANDALONE.

Once the application (instance) receives the SCCPRUNSTATEIND (SPRS_PRIMARY or SPRS_STANDALONE), normal data traffic through the primary SCCP instance can begin.

Note: The application also receives NOWPRIMARY/NOWBACKUP events from the HMI service. These events should be ignored for the purposes of starting data traffic towards the SCCP service. Only the SCCPRUNSTATEIND (SPRS_PRIMARY or SPRS_BACKUP) events should be honored for this purpose.

The following illustration shows the operations that occur at SCCP application startup:



SCCP normal operation

During normal operation, data traffic is transferred between the primary board and the primary application instance (context) only. This applies to both connectionless and connection-oriented service classes.

Connectionless data transfer functions issued to the backup board are discarded or returned to the caller if the return option is selected. Connection-oriented functions are refused (connection requests) or discarded (all others). Connection audit requests can be issued to both the primary and backup boards.

SCCP switchovers

Switchover processing is initiated when an application receives a SCCPRUNSTATEIND event with a new state that differs from the previous state of that board.

Single node application

For a single-node application that opens both boards, the application receives a SCCPRUNSTATEIND event (SPRS_PRIMARY) for the previously backup board and a SCCPRUNSTATEIND event (SPRS_BACKUP) for the previously primary board. These event indications can occur in either order, depending on how the switchover was initiated. For a dual-node architecture, the application instance receives only the new run state indication for its own board.

Connectionless traffic

For connectionless traffic, there is no special processing that the application must undertake other than to redirect its traffic to the new primary board. Connectionless messages that were in progress prior to the switchover may be lost, so the application must be able to recover from lost messages.

Connection-oriented traffic

For connection-oriented traffic, only confirmed connections are maintained by the SCCP layer across a switchover. Any transient connections (those in connecting state or in releasing state) are discarded. Resources associated with transient connections are recovered by protocol timers and the SCCP layer inactivity timers, transparently to the applications.

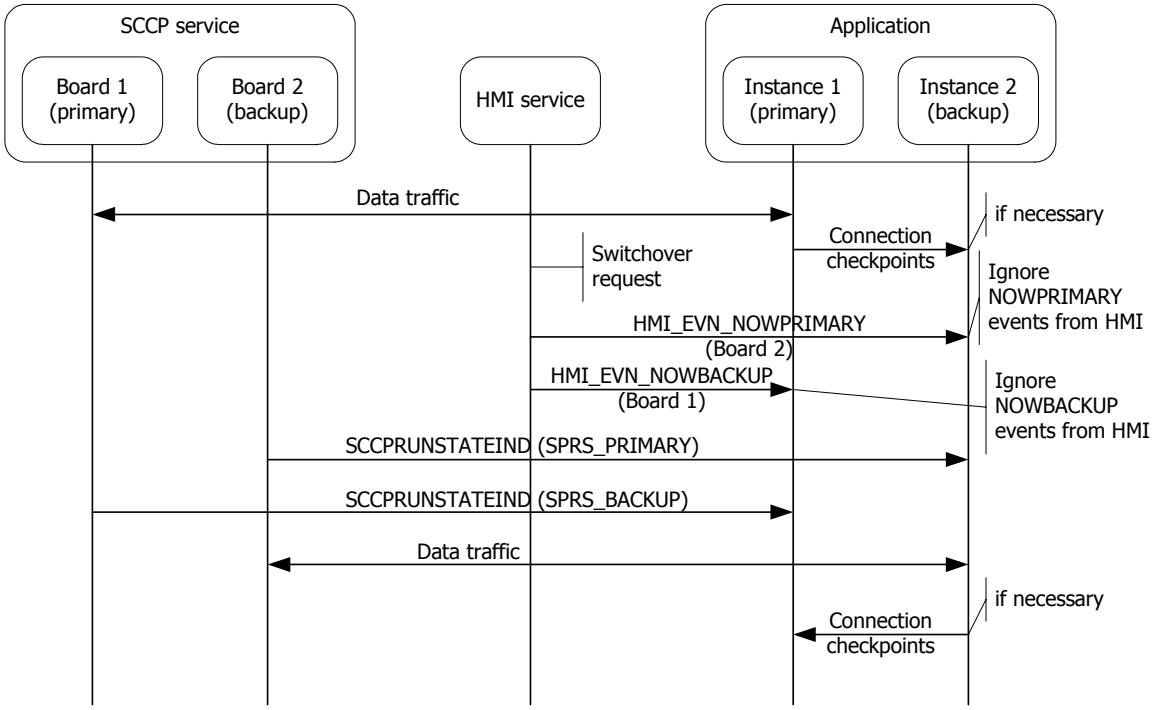
Therefore, a dual-node application employing connection-oriented services must implement some mechanism, such as sending checkpoint messages, to reflect the state of active connections from its primary instance to its backup instance. It is only necessary to checkpoint connection states when the connection is confirmed and released. For a single-node application that opens both boards, it may be necessary to examine connection tables and delete connections that were in the connecting or releasing states at the time of the switchover.

Timing windows

In some switchover cases, due to timing windows, it is possible that the SCCP layer view of which connections are active differs from the application view. To prevent the stranding of resources in the SCCP layer due to these timing windows, the application must implement the SCCP application inactivity timing facility and/or the connection auditing facility described in the *Dialogic® NaturalAccess™ SCCP Layer Developer's Reference Manual*.

Switchover processing

The following illustration shows the operations that occur during SS7 SCCP switchover processing:

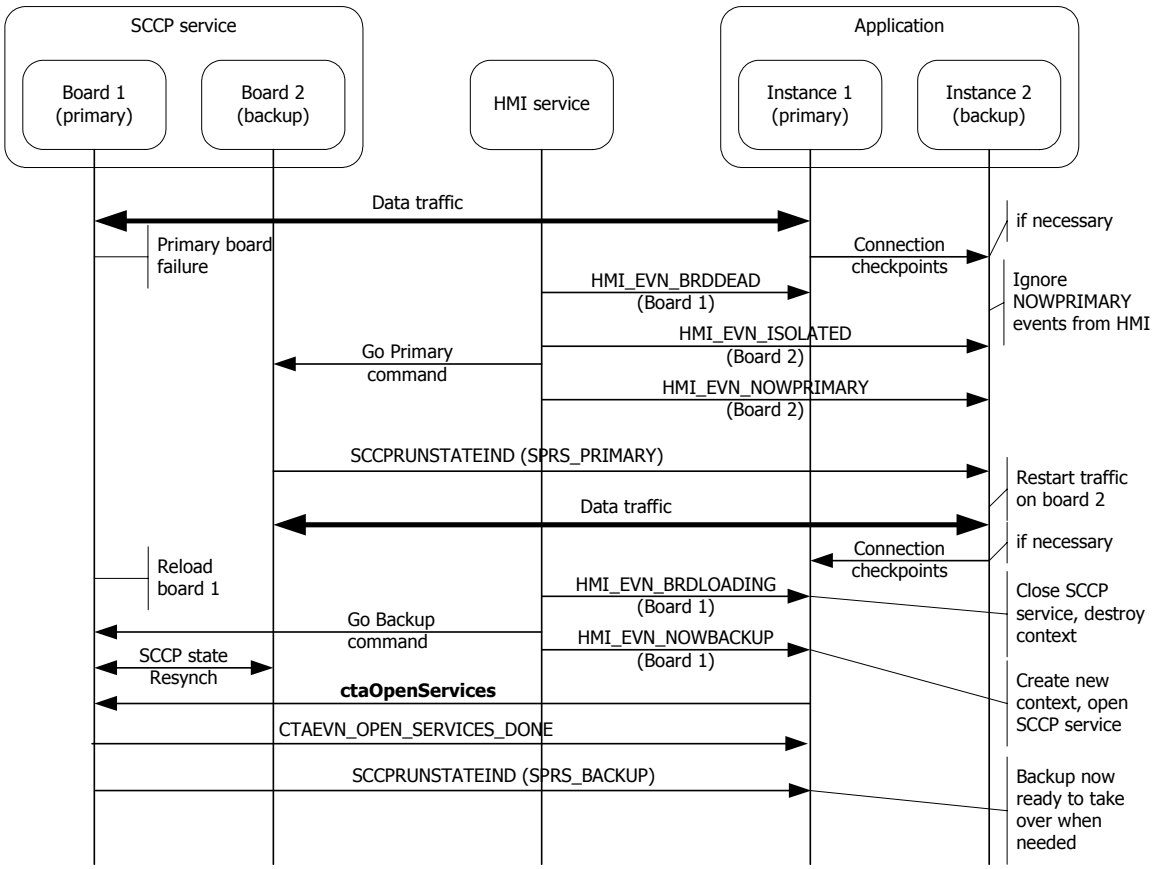


SCCP board failure and reload

A switchover can also be caused by a board failure (hardware or software). Switchover processing by the backup application instance transitioning to the primary state is similar to a planned switchover.

However, in a failure case, the failed board must be reloaded and placed into the backup state. The application must close the SCCP service and destroy the context associated with the failed board. Once the failed board is reloaded, the application (instance) must create a new context and re-open the SCCP service on the reloaded board.

When the board is reloaded and placed into the backup state by the system manager, the SCCP layers automatically re-synchronize their states so that the newly reloaded backup is again ready to take over in case of a failure of the primary. No application action is necessary to trigger this re-synchronization.



10 Setting up a redundant system

Board installation and cabling

Use the redundancy feature to enable the system to detect and recover from the failure of signaling links on a TX board, the failure of a signaling node, or the failure of the TX board itself.

In a redundant configuration, each pair of TX boards is connected through a private Ethernet connection. If other devices are connected to the private Ethernet link, avoid overloading the link. Packets can be lost between the redundant TX boards if the connection is overloaded.

Both TX boards of a redundant pair must be the same type of board (TX 4000 should be paired with another TX 4000; TX 5000 Series board should be paired with the same type of TX 5000 Series board).

This topic describes dual-node redundant signaling and single-node redundant signaling for the following types of configurations:

- TDM configuration
- IP network configuration

TDM configuration

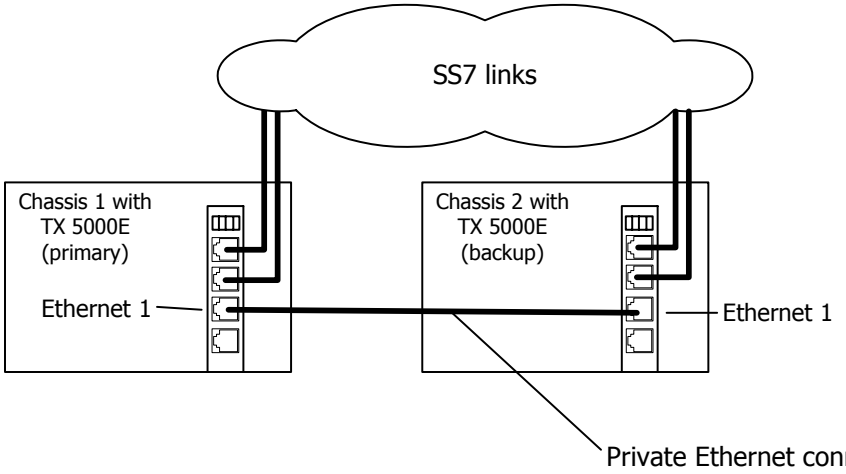
To connect a TX board to its redundant mate in a TDM configuration, use a Category 5 shielded twisted pair (STP) crossover cable. To connect two TX 5000 Series boards, use a crossover cable to connect Ethernet 3 on the primary board to Ethernet 3 on the backup board. To connect two TX 4000 boards, use a crossover cable, connect Ethernet 1 on the primary board to Ethernet 1 on the backup board.

You must create the IP interface using the ifcreate command in the *txconfig* utility. You must also specify the IP address of the TX board's redundant mate using the mate command in the *txconfig* utility. For more information, refer to the *Dialogic® NaturalAccess™ Signaling Software Configuration Manual*.

Dual-node redundant signaling server

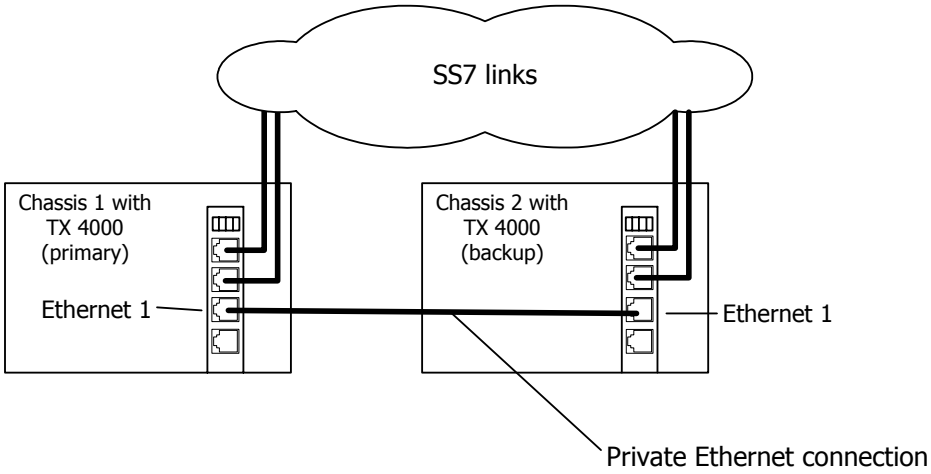
The following illustrations show how to set up two TX 5000E boards based on a dual-node redundant signaling server in a TDM configuration. The boards are located in two separate chassis to ensure board-level and system-level redundancy.

The following illustration shows a dual-node redundant signaling server for TX 5000E boards in a TDM configuration:

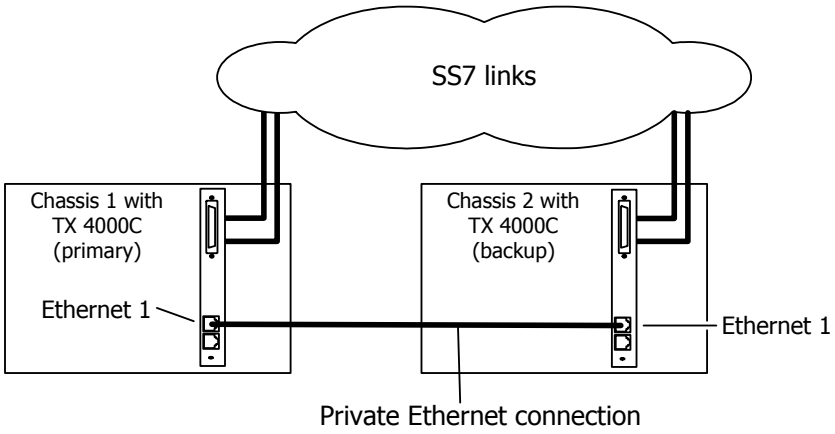


The following illustrations show how to set up two TX 4000 or TX 4000C boards based on a dual-node redundant signaling server in a TDM configuration. The boards are located in two separate chassis to ensure board-level and system-level redundancy.

The following illustration shows a dual-node redundant signaling server for TX 4000 boards:



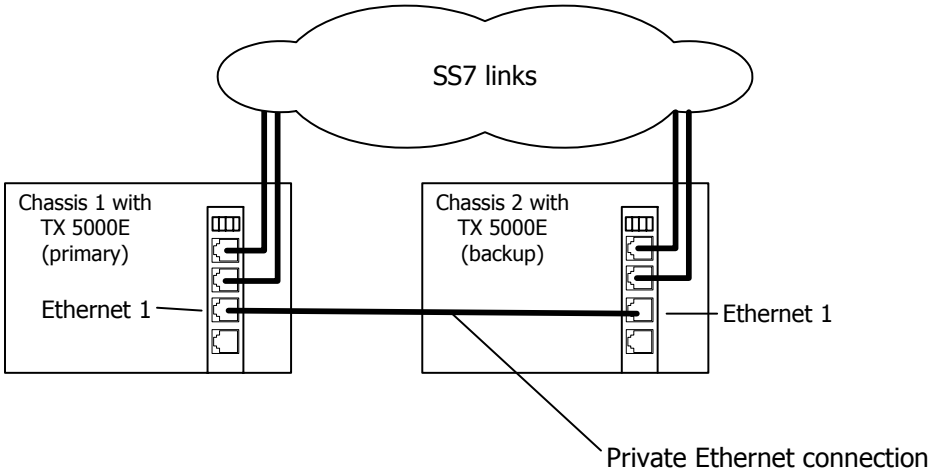
The following illustration shows a dual-node redundant signaling server for TX 4000C boards:



Single-node redundant signaling server model

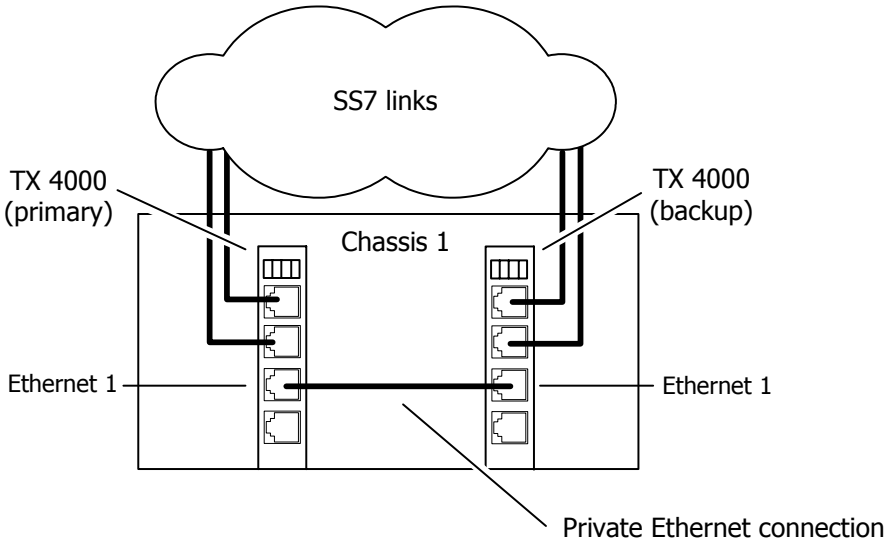
The following illustrations show how to set up two TX 5000E boards based on the single-node signaling server in a TDM configuration. The boards are located in the same chassis to ensure board-level redundancy.

The following illustration shows a single-node redundant signaling server for TX 5000E boards in a TDM configuration:

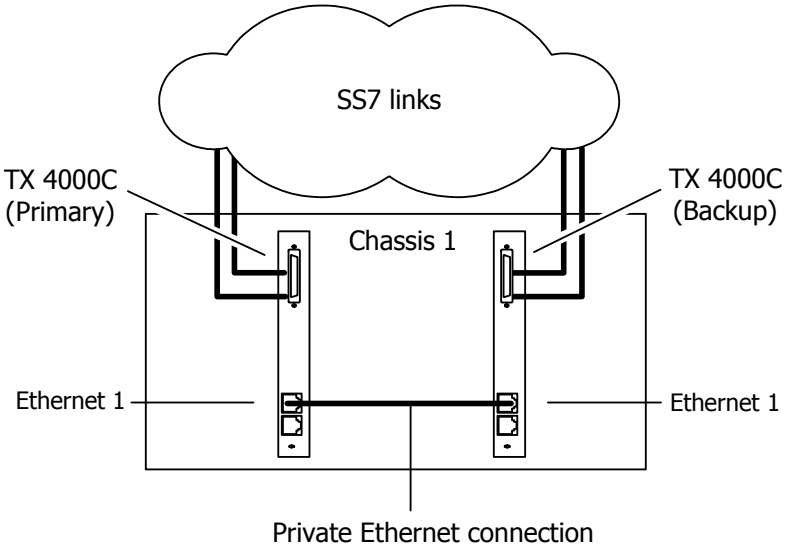


The following illustrations show how to set up two TX 4000 or TX 4000C boards based on the single-node signaling server in a TDM configuration. The boards are located in the same chassis to ensure board-level redundancy.

The following illustration shows a single-node redundant signaling server for TX 4000 boards:



The following illustration shows a single-node redundant signaling server for TX 4000C boards:



IP network configuration

To connect a TX board to its redundant mate in an IP network configuration, use a Category 5 shielded twisted pair (STP) crossover cable. To connect two TX 5000E boards for redundancy, use the crossover cable to connect Ethernet 3 on the primary board to Ethernet 3 on the backup board. To connect two TX 4000 boards for redundancy, use the crossover cable to connect Ethernet 1 on the primary board to Ethernet 1 on the backup board.

Using standard Ethernet cables, connect the remaining Ethernet connectors on both boards to the IP network. For a TX 5000E board, each board provides two Ethernet connectors that can be dedicated to SIGTRAN network access (Ethernet 1 and

Ethernet 2). For a TX 4000 board, each board provides a single Ethernet connector for SIGTRAN network access (Ethernet 2, since Ethernet 1 is used for redundancy).

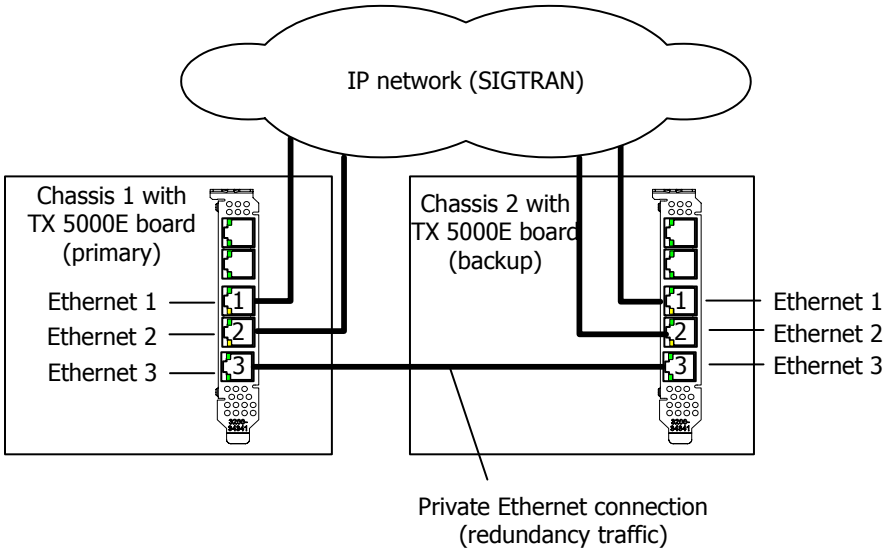
Use a private Ethernet link to connect the redundant boards to avoid loss or delay of vital checkpoint messages. Since TX 5000E boards provide three physical Ethernet connectors, it is possible to have a dedicated redundancy connection while still having redundant physical pathways from each board to the SIGTRAN network. TX 4000 boards provide two Ethernet connectors so if each board in the redundant pair requires multi-homing, you can use Ethernet 1 for both the redundant pathway and for SIGTRAN network access. In this configuration, the Ethernet 1 on each board is connected to what is shown as an IP network cloud in the illustrations that follow (just as the Ethernet 2 connectors are). Be aware that this greatly increases the chance of lost or delayed checkpoint messages which can result in the backup having outdated information.

You must create the IP interface using the ifcreate command in the *txconfig* utility. You must also specify the IP address of the TX board's redundant mate using the mate command in the *txconfig* utility. For more information, refer to the *Dialogic® NaturalAccess™ Signaling Software Configuration Manual*.

Dual-node redundant signaling server

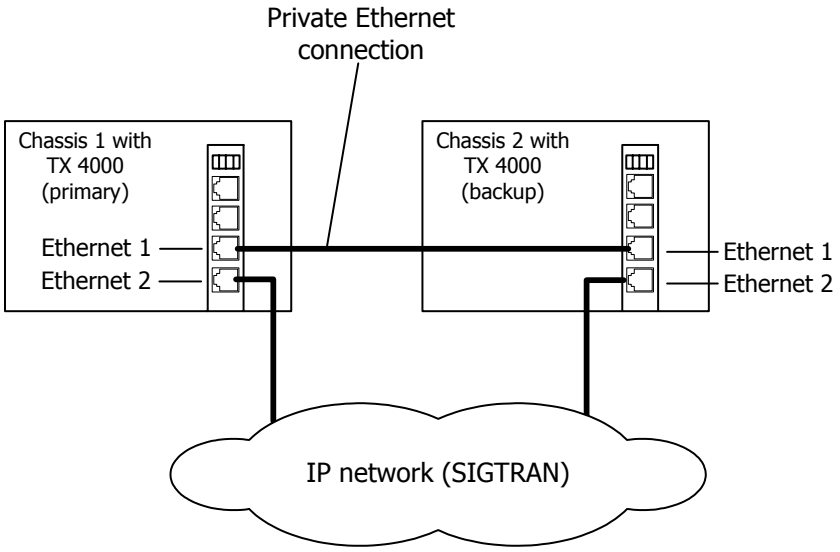
The following illustrations show how to set up two TX 5000E boards based on a dual-node redundant signaling server in an IP network configuration. The boards are located in two separate chassis to ensure board-level and system-level redundancy.

The following illustration shows a dual-node redundant signaling server for TX 5000E boards in an IP configuration:

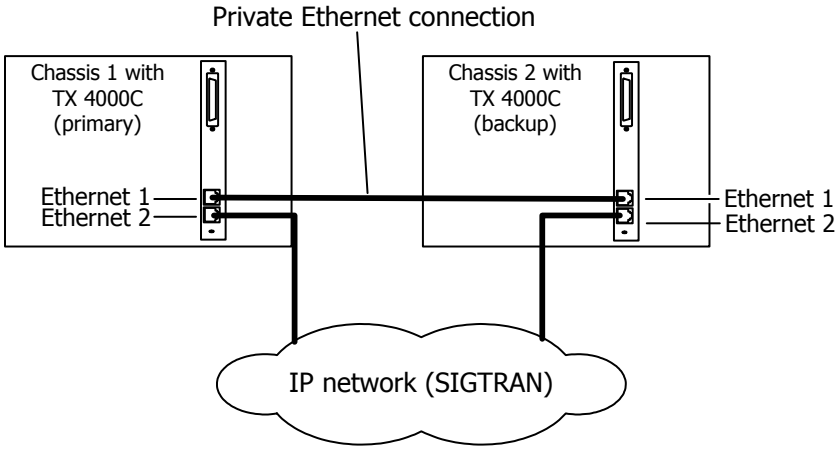


The following illustrations show how to set up two TX 4000 or TX 4000C boards based on a dual-node redundant signaling server in an IP network configuration. The boards are located in two separate chassis to ensure board-level and system-level redundancy.

The following illustration shows a dual-node redundant signaling server for TX 4000 boards:



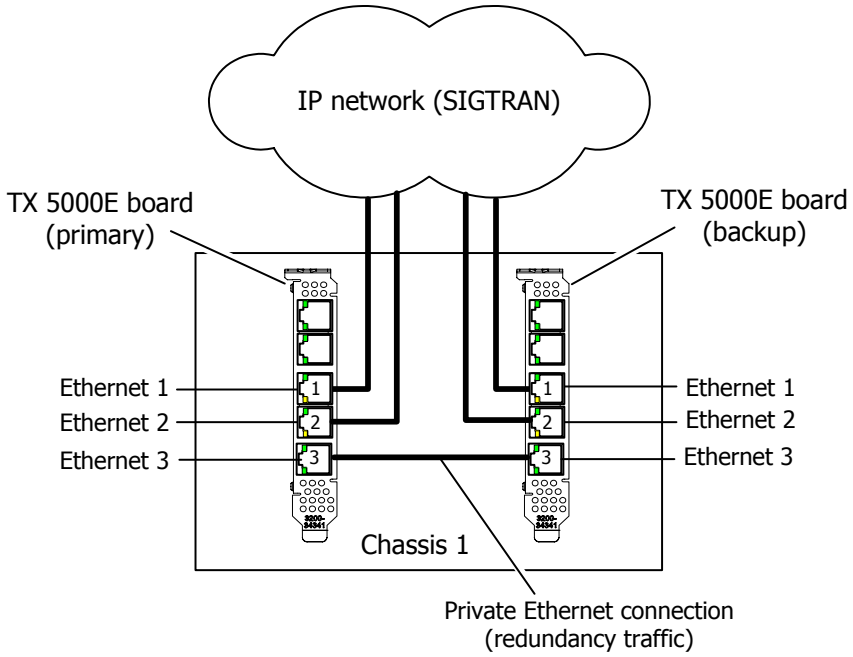
The following illustration shows a dual-node redundant signaling server for TX 4000C boards:



Single-node redundant signaling server

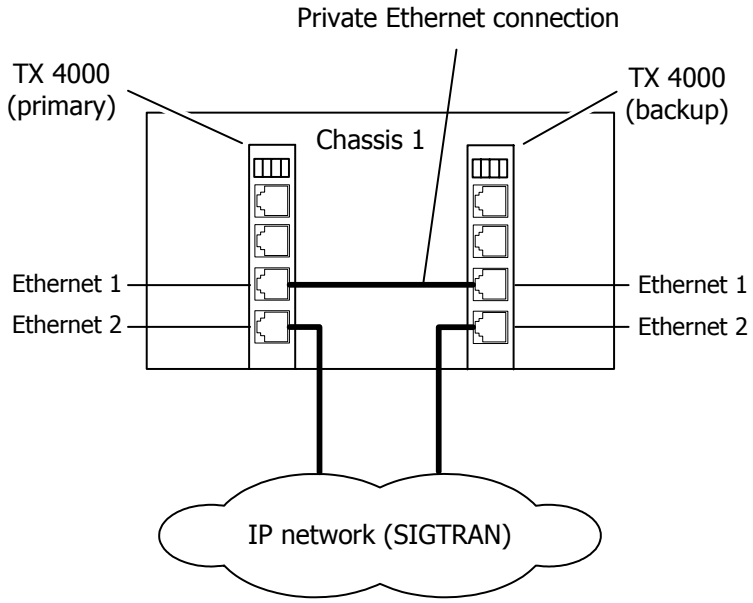
The following illustration shows how to set up two TX 5000E boards based on a single-node signaling server in an IP network configuration. The boards are located in the same chassis to ensure board-level redundancy.

The following illustration shows a single-node redundant signaling server for TX 5000E boards in an IP configuration:

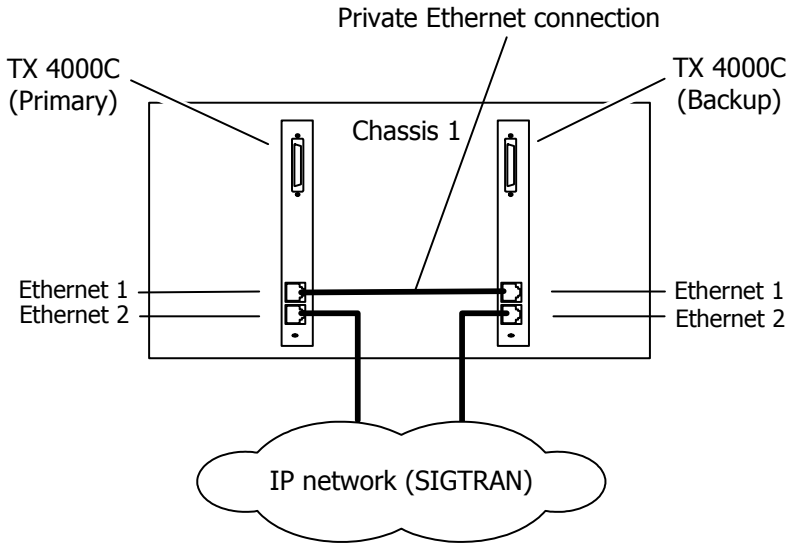


The following illustration shows how to set up two TX 4000 or TX 4000C boards based on a single-node signaling server in an IP network configuration. The boards are located in the same chassis to ensure board-level redundancy.

The following illustration shows a single-node redundant signaling server for TX 4000 boards:



The following illustration shows a single-node redundant signaling server for TX 4000C boards:



Configuring for redundant operation

To configure a system for redundant operation, modify the following components:

- HMI configuration
- *ss7load* script configuration
- MTP configuration over TDM
- M3UA and SCTP configuration over IP (SIGTRAN)
- ISUP, TUP, TCAP, and/or SCCP configuration

HMI configuration

The HMI service requires the *hmi.cfg* file to identify the boards to be monitored and other run time parameters. The HMI configuration file is located in the following directories:

Operating system	Directory
Windows	*\Program Files\Dialogic\tx\config\hmi.cfg*
UNIX	*/opt/dialogic/tx/etc/hmi.cfg*

Sample HMI configuration file

The sample HMI configuration file created by the software installation utility for Windows is shown in the following example. The UNIX file is identical to the Windows file except for the format of the path name for the download files.

Windows version

```
BOARD_NUMBER1     1
SS7LOAD_FILE1     "c:\\Progra~1\\Dialogic\\tx\\bin\\ss7load.bat"
BOARD_NUMBER2     2
SS7LOAD_FILE2     "c:\\Progra~1\\Dialogic\\tx\\bin\\ss7load2.bat"
END
```

UNIX version

```
BOARD_NUMBER1   1
SS7LOAD_FILE1   /opt/dialogic/tx/bin/ss7load
BOARD_NUMBER2   2
SS7LOAD_FILE2   /opt/dialogic/tx/bin/ss7load2
END
```

HMI configuration can be specified in any order. The BOARD_NUMBER***n*** and SS7LOAD_FILE***n*** can be repeated for up to eight boards:

Parameter name	Range	Default	Description
BOARD_NUMBER1	1 - 8	None	Board number to manage.
SS7LOAD_FILE1	N/A	None	File and path name of the batch file used to download the board. Ensure that the file is board specific.
BOARD_NUMBER2	1 - 8	None	Board number to manage.
SS7LOAD_FILE2	N/A	None	File and path name of the batch file used to download the board. Ensure that the file is board specific.

Note: HMI cannot pass any arguments (that is, the board number) to the *ss7load* script. If multiple boards are configured on a single node, each board must have its own *ss7load* script with an explicit board number. The default *ss7load* script is a generic script that accepts the board number as an argument and defaults to board number 1 when the board number is not specified.

Configuring port numbers for the Health Management service

The Health Management service uses one UDP port number for its functions - the same port for unsolicited events and for conversational requests. By default, port number 4801 is used for this purpose. Both the HMI service and the Health Management service libraries check for the appropriate service name/port assignments with the standard sockets **getservbyname** function before using the default values.

The service name is *hm_api*. For example, the following entry in the services file changes the UDP ports used to 1750:

```
hm_api      1750/udp      # Tx Series HM API service
```

This entry is read-only at startup time. To change these values, the HMI service and all applications using the Health Management service must be stopped and restarted.

ss7load script configuration

The *ss7load* script contains the commands necessary to download a board with the proper protocol tasks and configuration files. The *ss7load* script is located in the following directories:

Operating system	Directory
Windows	*\Program Files\Dialogic\tx\bin\ss7load.bat*
UNIX	*/opt/dialogic/tx/bin/ss7load*

For redundant configurations, the *ss7load* script must download the *txmon.elf* file. This is controlled by the value of the environment variable TXMODE. If TXMODE is set to redundant, then the ss7load script will load the *txmon.elf* file.

Note: For standalone configurations where the Health Management system is employed for board monitoring only, the *txmon.elf* file is required for standalone configurations.

The *ss7load* script is called by the HMI process, as shown in the HMI configuration file. In the sample HMI configuration file, there are two versions of the *ss7load* script file, *ss7load.bat* and *ss7load2.bat*. The first script loads board 1 and the second script loads board 2. It is important that both *ss7load* scripts are modified as shown earlier in this topic. If the *ss7load* script file names are changed, ensure that their references in the HMI configuration file are also changed.

MTP configuration over TDM

Specify the IP address of the local board using the ifcreate command and the IP address of the TX board's redundant mate using the mate command in the file read by the *txconfig* utility. This is typically specified in the *txcfg**n***.txt file, where ***n*** is the board number. For example:

```
#-------------------------------------------------------------------------
# Ethernet interface number 1:
ifcreate 1 192.168.1.1 255.255.0.0
#-------------------------------------------------------------------------|
# Set up this board's redundancy mate board address:
mate 192.168.1.2
```

For more information, refer to the *Dialogic® NaturalAccess™ Signaling Software Configuration Manual*.

A typical redundant configuration includes signaling links terminated on both boards in a mated pair. The MTP configuration for each board must include the links terminated on its own board and the links terminated on the mate board. The following example shows a simple MTP configuration file for each board of a mated pair with one link terminated on each board:

```
# Board 1 MTP Configuration          # Board 2 MTP Configuration
#-------------------------          #-------------------------
#Overall MTP3 Parameters             #Overall MTP3 Parameters
#-------------------------          #-------------------------

NODE_TYPE SP                        NODE_TYPE SP
. . .                               . . .
   <identical to board 2>              <identical to board 1>
. . .                               . . .
#-------------------------          #-------------------------
#Link 0                             #Link 0
#-------------------------          #-------------------------
LINK  0                             LINK  0
PORT  T1 # Board 1, TDM 1           PORT  R # Remote (board 1)
LINK_SET 1                          LINK_SET 1
ADJACENT_DPC 2.2.2 # Adjacent STP   ADJACENT_DPC 2.2.2 # Adjacent STP
LINK_SLC 0                          LINK_SLC 0
TIMER_T31  1                        TIMER_T31  1
LSSU_LEN 2                          LSSU_LEN 2         # ignored: remote
END                                 END
#-------------------------          #-------------------------
#Link 1                             #Link 1
#-------------------------          #-------------------------
LINK  1                             LINK  1
PORT  R # Remote (board 2)          PORT  T1 # Board 2, TDM 1
LINK_SET 1                          LINK_SET 1
ADJACENT_DPC 2.2.2 # Adjacent STP   ADJACENT_DPC 2.2.2 # Adjacent STP
LINK_SLC 1                          LINK_SLC 1
LSSU_LEN 2         # ignored: remote LSSU_LEN 2
END                                 END
#---------------------------------  #---------------------------------
#User Parameters (NSAP definition)  #User Parameters (NSAP definition)
#---------------------------------  #---------------------------------
. . .                               . . .
   <identical to board 2>              <identical to board 1>
. . .                               . . .
```

The locally terminated links are fully configured with physical port identifiers and all layer 2 and layer 3 parameters. The links terminated on the mate board contain no physical port identifiers or other layer 2 parameters; instead, they are specified as Remote (R).

Note: Each relative link number (link 0, for example) refers to the same link. On board 1, it is fully configured since it is terminated locally. On board 2, only the layer 3 parameters are specified because the link is not terminated on board 2 (layer 2 parameters can be specified in this case, but they have no affect).

Other sections of the MTP configuration, such as link sets, routes, and service access points are typically identical on both boards in the mated pair.

For more information on the MTP configuration, refer to the *Dialogic® NaturalAccess™ Signaling Software Configuration Manual.*

M3UA and SCTP configuration over IP (SIGTRAN)

Specify the IP address of the local board using the ifcreate command and the IP address of the TX board's redundant mate using the mate command in the file read by the *txconfig* utility. This is typically specified in the *ipcfg**n***.txt file, where ***n*** is the board number. For example:

```
#-------------------------------------------------------------------------
# Ethernet interface number 1 is part of the 10.*.*.* subnet:
ifcreate 1 192.168.1.1 255.255.0.0
#-------------------------------------------------------------------------
# Set up this board's redundancy mate board address:
mate 192.168.1.2
```

There are no specific M3UA or SCTP configuration requirements for redundant board configurations. Both boards in a mated pair typically have identical configurations.

For more information, refer to the *Dialogic® NaturalAccess™ Signaling Software Configuration Manual*.

ISUP, TUP, TCAP, and/or SCCP configuration

There are no specific ISUP, TUP, TCAP, or SCCP configuration requirements for redundant board configurations. Both boards in a mated pair typically have identical configurations.

For more information on the ISUP, TUP, TCAP, and SCCP configurations, refer to the *Dialogic® NaturalAccess™ Signaling Software Configuration Manual.*

Installing and running HMI in Windows

The Health Management Interface (HMI) runs as a service under Windows. After installation of the software, the HMI is installed as a Windows service in a default state of disabled.

Note: To use Hot Swap, the HMI service must be removed (hmi -remove) and re-installed with the -hsinstall option..

To start the HMI service, reboot the system or perform the following steps:

Step	Action
1	From the Control Panel, select the **Services** icon.
2	Choose the **Dialogic TX HMI** service.
3	Click **Start**.

On subsequent system boots, the service is started automatically.

Note: Before starting the HMI service, the HMI configuration file must be created or updated. Refer to *HMI configuration* on page 80.

Installing and running HMI in UNIX

In UNIX, the HMI service is named *hmid* and runs as a daemon process.

It can be started manually from a command line prompt or started at boot time from within a startup script.

The *hmid* daemon has no command line parameters and generates no output messages.

Bringing up a redundant system

Once configuration is complete and the HMI service has been started, start the redundant system by completing the following steps:

Step	Action
1	Start the txalarm utility.
2	Load the first board and set it to the primary state. Refer to *Loading and setting board states* on page 86 for information.
3	Load the second board and set it to the backup state.
4	Start application traffic.

Starting txalarm

The *txalarm* utility is the primary source for status information regarding communication problems between boards in a redundant configuration.

Before starting the redundant system, start the *txalarm* utility to monitor the startup and ensure that the system is working correctly. If you are running a dual node system, start *txalarm* on both systems.

To run *txalarm* from an MS-DOS prompt (Windows) or a UNIX shell, type:

```
txalarm -f alarm.log
```

Messages display when the boards are first loaded. The `-f` option saves the output to a file for later reference.

Loading and setting board states

These steps can be performed with the redundancy manager (RMG) demonstration program or with an application that uses the Health Management service. In this topic, it is assumed that the RMG demonstration program is used.

The RMG demonstration program (*rmg.exe* [Windows] or *rmg* [UNIX]) is located in the following directories:

Operating system	Directory
Windows	*\Program Files\Dialogic\tx\bin*
UNIX	*/opt/dialogic/tx/bin/*

For more information, refer to the *RMG demonstration program overview* on page 93.

RMG on a single node system

In a single node system, both boards of the redundant set are in the same machine. A separate instance of the RMG demonstration program must be invoked for each board of the redundant set.

One board is arbitrarily designated as node 1 of the redundant set. The other board is designated as node 2 of the redundant set. In this example, board 1 is node 1 and board 2 is node 2. Each RMG must also designate a UDP port over which to communicate with the other RMG. By default, this port number is 1700. However, when the boards are on the same machine they must use different ports. For this example, the RMG for board 1 uses port 1700 and the RMG for board 2 uses port 1701.

For the first board, type the following command from an MS-DOS prompt:

```
rmg -b 1 -n 1 -p 1701
```

For this example, board 1 (-b 1) is designated as node 1 (-n 1). Its local port is 1700 (default), and the port number of its mate is 1701 (-p 1701).

For the second board, type the following command:

```
rmg -b 2 -n 2 -l 1701
```

For this example, board 2 (-b 2) is designated as node 2 (-n 2). Its local port is 1701 (-l 1701), and the port number of its mate is 1700 (default).

RMG on a dual node system

To use RMG in a dual node system, both machines must have an IP network connection to the other machine. This must be a different connection from the Ethernet crossover that links the two TX boards.

In a dual node system, each board of the redundant set is in a different machine. A separate instance of RMG must be invoked for each board of the redundant set.

One board is arbitrarily designated as node 1 of the redundant set. The other board is designated as node 2 of the redundant set. In this example, board 1 of machine A is node 1 and board 1 of machine B is node 2. To communicate with its mate, each RMG must also designate the IP address and UDP port of its mate. Since the boards are on separate machines, they can both use the default UDP port number, 1700.

First board, machine A

For the first board, on machine A, type the following from an MS-DOS prompt:

```
rmg -b 1 -n 1 -m 1.1.1.2
```

For this example, board 1 (`-b 1`), of machine A, is designated as node 1 (`-n 1`). Its partner is machine B, whose IP address is 1.1.1.2. Its local port is 1700 (default), and the port number of its mate is 1700 (default).

Alternatively, the host name of machine B is specified with the `-m` command line option. Under Windows (with Microsoft Networking) or under UNIX, the host name can be used.

Second board, machine B

For the second board, on machine B, type:

```
rmg -b 1 -n 2 -m 1.1.1.1
```

For this example, board 1 (`-b 1`) of machine B is designated as node 2 (`-n 2`). Its partner is machine A, whose IP address is 1.1.1.1. Its local port is 1700 (default), and the port number of its mate if 1700 (default).

Alternatively, the host name of machine A is specified with the `-m` command line option. Under Windows (with Microsoft Networking) or under UNIX, the host name can be used.

RMG startup

Each RMG, on startup, attempts to load its designated board if the board has not already been loaded. The load script used is defined in the HMI configuration file. Refer to *HMI configuration* on page 80 for more information.

The first node that starts RMG becomes the primary node. For this example, it is assumed that node 1 starts RMG first, which loads the board on node 1 first, and then goes to the primary state. The RMG for node 1 displays:

```
Board 1, Node 1  Board Loading
Board 1, Node 1  Now Starting
Board 1, Node 1  Board Isolated
Board 1, Node 1  Board Connected
Board 1, Node 1  Now Primary
```

It is assumed that node 2 starts RMG second. Node 2 goes to the backup state. The RMG for node 2 displays:

```
Board 1, Node 2  Board Loading
Board 1, Node 2  Now Starting
Board 1, Node 2  Board Isolated
Board 1, Node 2  Board Connected
Board 1, Node 2  Now Backup
```

If...	Then the...	For more information, refer to...
Both RMG applications display Now Primary	RMG applications are not communicating.	Troubleshooting RMG communication
The board connected event is not displayed	Boards are not communicating	Troubleshooting board communication

Troubleshooting RMG communication

If the RMG demonstration programs are not communicating, both of the boards go to the primary state (the Now Primary message displays for both boards). If this occurs, check the following items:

- Physical IP network connection for each node.
- IP addresses (or host names) used when RMG is started.
- The local UDP port number of one RMG application matches the remote UDP port number of the other RMG application.

Troubleshooting board communication

If a board connected event is not displayed by either RMG, there is a problem with the board to board connection.

Use the *txalarm* utility to determine the board communication state. During a successful system startup, the inter-board communication status should become connected, as shown in the following sample alarm output:

```
<05/14/1999 14:53:14> txmon  1    19745 Initialization complete
<05/14/1999 14:53:14> mtp    1        1 Configuring MTP Layer 1
. . .
<05/14/1999 14:53:14> mtp    1        1 MTP3: Ready...
<05/14/1999 14:53:14> txmon  1    19746 Task [mtp ] registered
<05/14/1999 14:53:14> txmon  1    19748 Mate board found at IP address
                                         64.0.21.132
<05/14/1999 14:53:15> mtp    1        1 MTP3 Connected
<05/14/1999 14:53:15> isup   1        1 Registering ISUP Layer
<05/14/1999 14:53:15> isup   1        1 ISUP: Ready...
<05/14/1999 14:53:15> txmon  1    19746 Task [isup ] registered
. . .
<05/14/1999 14:53:17> mtp    1    18179 MTP3 Link 0 Up
```

If the boards cannot communicate after being downloaded, they remain isolated. During isolation, the signaling links terminated on the backup cannot be brought into service, and the backup board will not correctly reflect the state of the network. The most likely causes of isolation during turn-up of a new installation are:

- Ethernet ports on the mated boards are not properly connected with a crossover cable.
- *txmon* task was not downloaded when the board was loaded.
- Mate IP addresses are not configured properly for both boards.

Once the boards are properly connected, enter the status (S) command to each RMG. Each RMG displays the following status:

```
RMG Board n Status
--------------------------
State :ACTIVE
Network Status : UNKNOWN
Heartbeats Sent: nnn       Received: nnn
HMI Board n Status
-----------------------
Heartbeats Sent: nnn        Received: nnn
Link State : Connected
Network State : Not Reported
mtp State  = Primary
isup State = Primary
```

Confirm that the state is active and the link state is connected.

Starting data traffic

Once boards are loaded and successfully communicating, normal data traffic can begin.

If links are not established and/or data traffic is not successful, there may be a problem with the redundant MTP or SIGTRAN configuration.

Checking link status

Check the MTP link status to determine if the links are up. Use the *mtpmgr* application to determine the link status. MTP configuration provides a sample MTP configuration for each board having a single link to a third node at point code 2.2.2. Performing a **status link *** command for Node 1 for this configuration produces the following results:

```
Num    Name      MTP3 State       MTP2 Hi State     Low State
0      T1        ACTIVE           ENABLED           IN_SERVICE
1      R         ACTIVE           REMOTE
```

Performing the same command for Node 2 produces the following results:

```
Num    Name      MTP3 State       MTP2 Hi State     Low State
0      R         ACTIVE           REMOTE
1      T1        ACTIVE           ENABLED           IN_SERVICE
```

If the links are expected to be enabled and active, but are not in this state, check the MTP configuration.

Checking the MTP configuration

Ensure that the proper MTP configuration file is called from the proper *ss7load* script listed in the HMI configuration file. Also verify that links are configured with valid port types on the board they are physically terminated on and are configured as remote (R) on the mate board. For more information, refer to *MTP configuration over TDM* on page 82.

If the port type appears to be configured correctly but the links do not come into service, other problems may exist. Refer to the *Dialogic® NaturalAccess™ Signaling Software Configuration Manual* for information on troubleshooting MTP configurations.

Checking the association status

In a SIGTRAN configuration, check the SIGTRAN association status to determine if associations were established and M3UA management messages were exchanged correctly. To check the SIGTRAN association status, use the status psp 1 command, and check the values for State and ASP State in the output. For example:

```
m3uamgr[1]>status psp 1

=====================PSP 1 Association Status====================
AssocId = 0       State = ACTIVE  ASP State = ACTV    Inhibit = NO

=================Active PSs (4)=================
        PsId   = 3
        PsId   = 4
        PsId   = 1
        PsId   = 2

=================Registered PSs (0)==============
        None

====================Current PSP 1 Confuration=================
pspType       = IPSP    ipspMode      = DBL END dynRegAllow   = YES
loadShareMode = RNDRBN  nwkAppIncl    = NO      rxTxAspId     = NO
selfAspId     = 0       nwkId         = 1       PriDestAddr   = 10.51.1.185
DestPort      = 2905    locOutStrms   = 2
```

The following table describes the association states:

Association state	Description
ACTIVE	An association is established.
DOWN	No association is established. If M3UA is configured as an ASP or IPSP client, there is probably a configuration or connectivity problem. For information, refer to *Checking the SIGTRAN configuration* on page 90.

The following table describes the ASP states:

ASP state	Description
ACTV	ASPAC messages were exchanged, and traffic can flow.
DOWN	No M3UA ASP messages were exchanged. This should only occur if the association State is also DOWN.
INACTV	ASPUP messages were exchanged, but not ASPAC messages. This can occur if an upper layer has not yet bound to M3UA, or if this is the backup node. The ASPAC messages will be sent as soon as an upper layer binds to M3UA or the node becomes the primary node, or both.

Checking the SIGTRAN configuration

Ensure that the proper M3UA and SCTP configuration files are called from the proper ss7load script listed in the HMI configuration file. Also verify that the primary destination address specified for each PSP is correct. For more information, refer to *M3UA and SCTP configuration over IP (SIGTRAN)* on page 84.

If the configurations seem correct, verify that you have connectivity from each board to the remote nodes. Use the *cpcon* utility to send pings from each board to the destination addresses.

If the configuration and connectivity seem OK, other problems may exist. Refer to the *Dialogic® NaturalAccess™ Signaling Software Configuration Manual* for more information about troubleshooting SIGTRAN configurations.

ISUP testing

Once both boards of the redundant set are loaded and communicating and links are established, ISUP tests can take place using:

- Applications
- *orig* and *term* demonstration programs (refer to the *Dialogic® NaturalAccess™ ISUP Layer Developer's Reference Manual*)
- *isupdemo* demonstration program

TCAP testing

Once both boards are loaded and communicating and the links are established, TCAP tests can take place using:

- Applications
- *find800* demonstration program (refer to the *Dialogic® NaturalAccess™ TCAP Layer Developer's Reference Manual*)
- *tcapdemo* demonstration program

TUP testing

Once both boards of the redundant set are loaded and communicating and links are established, TUP tests can take place using:

- Applications
- *tuporig* and *tupterm* demonstration programs (refer to the *Dialogic® NaturalAccess™ TUP Layer Developer's Reference Manual)*
- *tupdemo* demonstration program

SCCP testing

Once both boards are loaded and communicating and the links are established, SCCP tests can take place using:

- Applications
- *sccpdemo* demonstration program (refer to the *Dialogic® NaturalAccess™ SCCP Layer Developer's Reference Manual*).

11 RMG demonstration program

RMG demonstration program overview

The redundancy manager (RMG) demonstration program is a sample management application for controlling a redundant board pair with the Health Management service. Each instance of the RMG application controls one member of a board pair and communicates with a peer RMG process that controls the mate board.

Note: The RMG demonstration program is provided solely as a sample application for illustrating control of a redundant board-pair through the Health Management service and as an aid for prototyping redundant configurations. It is not guaranteed to be complete or failure resilient and is not suitable for live system deployment.

The RMG program operates in either a single-node or dual-node configuration. One instance of RMG is run for each board. The RMG demonstration program communicates with its mate RMG demonstration program through UDP/IP using the sockets interface (even when both processes reside on a single node). Together, the RMG demonstration programs implement the failure detection and recovery policies recommended for a redundant configuration.

RMG also provides a command line interface for issuing Health Management service commands (load a board, halt a board, retrieve board status) and switching control between the primary and backup boards.

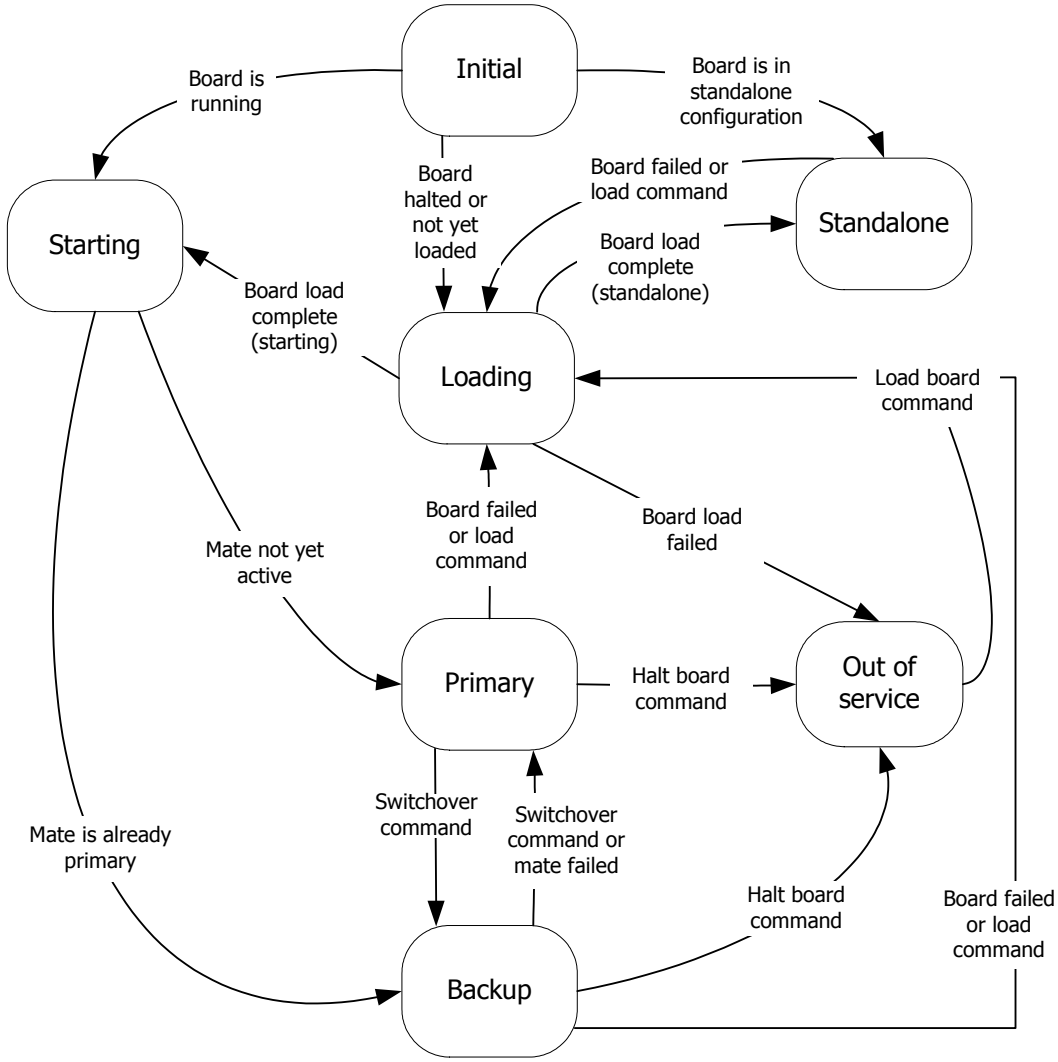
RMG requires TCP/IP networking and a sockets implementation (Windows Sockets version 1.1 or later for Windows, stands BSD sockets library for UNIX). For dual-node configurations, a suitable IP connection between nodes, such as a local area network, is required.

RMG can also be used in a standalone configuration (without the mate process) for detecting and recovering from board failures without user intervention.

RMG state model

The behavior of each signaling node, as implemented through the RMG demonstration program, is modeled as a finite state machine where the state of each node is determined by external events such as board failures, signaling node failures, and user commands. The RMG state model is shown in the following illustration.

Note: Some transient states and some events/transitions are not shown.



The following table describes the RMG states:

State name	Description
Initial	Initial state upon starting RMG process; determining if board has already been loaded.
Loading	Board is being downloaded.
Starting	Board is loaded; determining whether mate node is already active or not.
Standalone	Board is in standalone (non-redundant) configuration.
Primary	Board is in primary mode.
Backup	Board is in backup mode, monitoring status of primary board.
Out of service	Board has failed and attempt to reload it has failed; or, halt command received; manual intervention is required to restore board.

RMG initialization

The goal of the initialization phase is to independently start and restart signaling nodes. This results in a synchronized system (one in which both nodes agree on which is the primary and which is the backup) that restores signaling functionality as quickly as possible.

During initialization, an RMG process contacts its mate to determine if the mate is already primary. If no response is received or a communication error occurs, the RMG process delays for a short period and retries. If the retry is unsuccessful (or the mate determines it is the backup node), the restarting board becomes the primary board. The delay and retry is necessary to avoid having both nodes initialize simultaneously, unable to contact each other, and both become primary.

To resolve startup glare, where both nodes initialize, each RMG process is assigned a node number (1 or 2). The lower numbered node (node 1) becomes the primary node and the higher numbered node becomes the backup when startup glare is detected.

RMG failure detection and recovery

To facilitate failure detection and recovery, the primary RMG process monitors the board through the Health Management service. The primary RMG periodically sends heartbeat messages to the backup, allowing the backup to monitor the primary's status.

When the primary RMG process detects board failure or a reload or halt command is received, it initiates failure recovery by negotiating a switchover to the backup board. If possible, the failed board is reloaded and brought back into service as the backup board (unless a halt command was received, in which case the board is halted and remains out of service).

The RMG process (both primary and backup) also supports a planned changeover command that causes the primary and backup boards to switch roles.

To detect a failure of the primary RMG process or signaling node, the backup RMG continuously monitors for the receipt of heartbeat messages from the primary. If no heartbeat messages are received for five consecutive heartbeat periods, the backup initiates its own recovery and switches to primary mode.

Running the RMG demonstration program

RMG is started from a Windows command line console or a UNIX command line prompt:

```
RMG [-b board] [-l loc_port] [-m mate_addr] [-n node] [-p remote_port] [-t]
```

All run time parameters are optional and are defined in the following table.

Note: Each instance of RMG monitors and controls a single board.

Parameter	Description	Default value
-b ***board***	Board number to monitor on this node.	1
-l ***loc_port***	Local UDP port for this process to attach to.	1700
-m ***mate_addr***	IP address (in q.x.y.z dotted notation) or host name of the mate RMG process.	None. If omitted, it is assumed that the mate RMG process exists on the same node.
-n ***node***	Node number [1..2] assigned to this RMG process for resolving startup glare.	1
-p ***remote_ port***	UDP port where the mate RMG process can be found.	1700 If two RMG processes are executed on the same node, they must be assigned different UDP port numbers.
-q	Requires operator quit command before RMG exits.	Automatic exit on certain failure conditions.
-t	Enables tracing to the Natural Access server (*ctdaemon*).	No tracing.

Once running, the RMG process displays error messages and status change messages in the console window where it was started. The following example shows sample output from the RMG process:

```
host prompt> rmg -b2 -m node1 -n2 -t
rmg: Redundancy manager version 2.0 Sep 30 2008
Node: 2, Board 2: Board Halted
Node: 2, Board 2: Board Loading
Node: 2, Board 2: Now Starting
Node: 2, Board 2: Board Isolated
Node: 2, Board 2: Now Primary

RMG>
```

RMG supported commands

The RMG demonstration program includes a command line user interface for issuing HMI commands.

RMG does not automatically issue a prompt unless the user presses **Enter** to prevent scrambling messages being displayed. Enter user commands at any time, with or without the prompt.

The following table describes the RMG commands and their abbreviations:

Command	Quick command	Description
Status	**S**	Displays current board status and statistics.
Change	**C**	Swaps current primary and backup, if possible.
Halt	**H**	Halts the board, taking it out of service.
Load	**L**	Reloads the board.
Reset	**R**	Resets HMI.
Quit	**Q**	Quits this process without disturbing the board.
Help	**?**	Displays a list of available commands.

Tracing RMG events

Tracing events processed by the RMG demonstration program can be enabled by starting the Natural Access Server (*ctdaemon*) and running RMG with the `-t` option. This can be helpful in understanding the sequence of events in certain scenarios.

Configuring and starting the Natural Access Server (*ctdaemon*) is described in the *Natural Access Developer's Reference Manual.*

The following example shows sample trace output from the Natural Access Server (*ctdaemon*) when RMG is run with tracing enabled:

```
CT Access Daemon V.5 (Mar  4 1999)
ctdaemon: Configuration file './cta.cfg':
      [ctasys] section loaded.
ctdaemon: Configuration file './cta.cfg':
      [ctapar] section loaded.
ctdaemon> MESG: Thu May 13 10:31:10 1999
  | pid=6a tid=75 ctahd=80010002 (RMGCMD) uid=0 tag=4003 sev=0
  | DEBUG: RMG Controller FSM started
MESG: Thu May 13 10:31:10 1999
  | pid=6a tid=75 ctahd=80010002 (RMGCMD) uid=0 tag=4003 sev=0
  |  RMGC State: INITIAL       Event: Board Halted
MESG: Thu May 13 10:31:10 1999
  | pid=6a tid=75 ctahd=80010002 (RMGCMD) uid=0 tag=4003 sev=0
  |  RMGC State: LOADING       Event: Board Loading
MESG: Thu May 13 10:31:14 1999
  | pid=6a tid=75 ctahd=80010002 (RMGCMD) uid=0 tag=4003 sev=0
  |  RMGC State: LOADING       Event: Now Starting
MESG: Thu May 13 10:31:14 1999
  | pid=6a tid=75 ctahd=80010002 (RMGCMD) uid=0 tag=4003 sev=0
  |  RMGC State: STARTING      Event: Board Isolated
MESG: Thu May 13 10:31:17 1999
  | pid=6a tid=75 ctahd=80010002 (RMGCMD) uid=0 tag=4003 sev=0
  |  RMGC State: STARTING      Event: Timer_T1
MESG: Thu May 13 10:31:20 1999
  | pid=6a tid=75 ctahd=80010002 (RMGCMD) uid=0 tag=4003 sev=0
  |  RMGC State: STARTING      Event: Timer_T1
MESG: Thu May 13 10:31:20 1999
  | pid=6a tid=75 ctahd=80010002 (RMGCMD) uid=0 tag=4003 sev=0
  |  RMGC State: ACTIVE        Event: Now Primary
```

12 ISUP demonstration program

ISUP demonstration program overview

The ISUP demonstration program, *isupdemo*, is a multi-threaded program that uses the redundancy features of the SS7 ISUP layer and the Health Management service. It is a skeletal implementation of a toll switch with a user interface for placing and receiving test calls and managing circuits.

isupdemo data structures

The following data structures are called by *isupdemo*:

Structure	Description

ChkPntMsg	The checkpoint message structure transfers circuit state information from the primary application to the backup application:
	`typedef struct checkPointMsg`
	`{`
	`  U32  msgId;`
	`  CirId cirId;  /* circuit ID of indicated circuit */`
	`  CirId mateId;  /* circuit ID of mate circuit  */`
	`  U8  transient;  /* transient state indicator  */`
	`  U8  callState;  /* call processing state  */`
	`  U8  blkState;  /* circuit blocking state  */`
	`} ChkPntMsg;`

Event	The event structure passes information between threads by passing a pointer to an event in the buffer member of a CTA_EVENT structure:
	`typedef struct`
	`{`
	`  IsupRcvInfoBlk info;  /* ISUP receive information block  */`
	`  SiAllSdus  sdu;  /* union of all ISUP event structures */`
	`  ChkPntMsg  chkPntMsg;  /* checkpoint message  */`
	`} Event;`

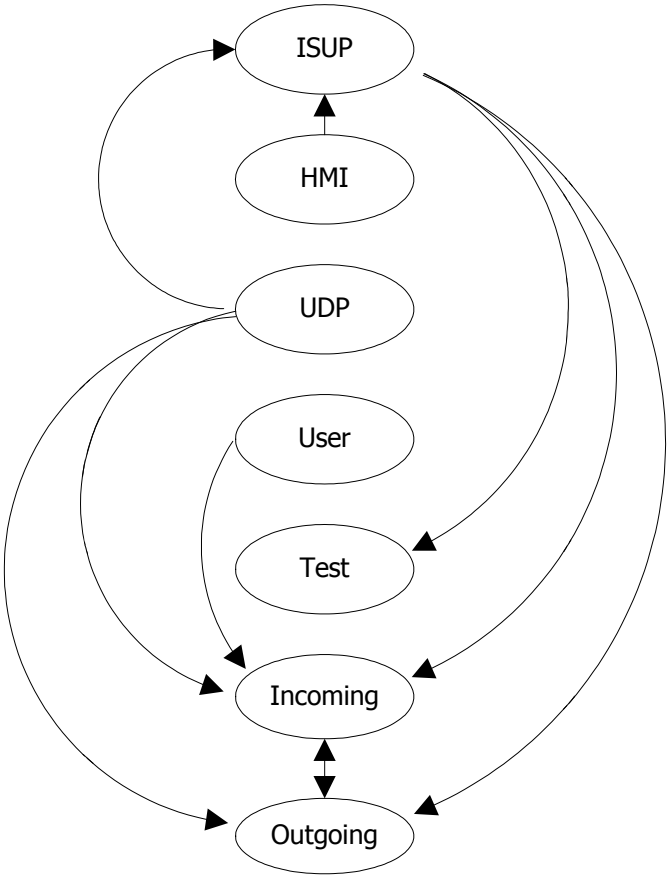
Circuit	The circuit control structure maintains information required by the application to control a particular circuit:
	`typedef struct circuit`
	`{`
	`  S16  state;  /* thread state  */`
	`  CTAQUEUEHD ctaQueue;  /* CTA queue for receiving events  */`
	`  CTAHD  ctaHndl;  /* CTA handle for this thread  */`
	`  CirId  cirId;  /* circuit ID of this circuit  */`
	`  CirId  mateId;  /* circuit ID for mate circuit  */`
	`  SuId  suId;  /* service user ID  */`
	`  SiInstId  suInstId;  /* service user instance ID  */`
	`  SiInstId  spInstId;  /* service provider instance ID  */`
	`  U8  callState;  /* call processing state  */`
	`  U8  blkState;  /* circuit blocking state  */`
	`  U8  transient;  /* transient state indicator  */`
	`} Circuit;`

isupdemo threads

The following threads comprise *isupdemo*:

Thread	Description
Main	Parses command line arguments, initializes global data, and starts all other threads.
User	Accepts input from the keyboard and parses the input. Depending upon the command entered by the user, this results in either an event to a circuit thread or a call to NaturalAccess™ ISUP.
Test	Receives events regarding a test call from a circuit thread resulting in a printed message indicating the type of event that was received.
UDP	Receives checkpoint messages from the mate application, which result in checkpoint events generated to circuit threads. It also receives the application ready message from the mate application, resulting in an application ready event sent to the ISUP thread.
HMI	Receives Natural Access events from the Health Management service and translates these events into internal events passed to the ISUP thread.
ISUP	Receives Natural Access events from NaturalAccess™ ISUP and translates these events into internal events routed to the appropriate incoming, outgoing, and test threads.
Incoming circuit	Receives events from the ISUP thread and exchanges events with its mate outgoing thread.
Outgoing circuit	Receives events from the ISUP thread and exchanges events with its mate incoming thread.

The following illustration shows the inter-thread communications in the ISUP demonstration program:



ISUP events

Threads use Natural Access to pass events between themselves. The Natural Access event ID values used to identify these events are:

- ISUP to circuit
- Circuit to circuit
- UDP to circuit
- UDP to ISUP
- HMI to ISUP

ISUP to circuit

The following event ID values pass information from the ISUP thread to circuit threads. They also pass information from the user thread to circuit threads.

Event	Description
IAM_MSG	Initial address message was received for this circuit.
ACM_MSG	Address complete message was received for this circuit.
ANM_MSG	Answer message was received for this circuit.
REL_MSG	Release message was received for this circuit.
RLC_MSG	Release complete message was received for this circuit.
RSC_MSG	Reset message was received for this circuit.
BLO_MSG	Blocking message was received for this circuit.
BLA_MSG	Blocking acknowledgement message was received for this circuit.
UBL_MSG	Unblocking message was received for this circuit.
UBA_MSG	Unblocking acknowledgement message was received for this circuit.
INF_MSG	Information message was received for this circuit.
INR_MSG	Information request message was received for this circuit.
CON_MSG	Connect message was received for this circuit.
CPG_MSG	Call progress message was received for this circuit.
SUS_MSG	Suspend message was received for this circuit.
RES_MSG	Resume message was received for this circuit.
SAM_MSG	Subsequent address message was received for this circuit.
CGB_MSG	Circuit group blocking message was received for this circuit.
CGU_MSG	Circuit group unblocking message was received for this circuit.
IDLE_EVT	Circuit should immediately transition to the idle state with no ISUP interaction.

Circuit to circuit

The following event ID values pass information between circuit threads:

Event	Description
IAM_EVT	Initial address message was received for this circuit's mate.
ACM_EVT	Address complete message was received for this circuit's mate.
ANM_EVT	Answer message was received for this circuit's mate.
REL_EVT	Release message was received for this circuit's mate.
RLC_EVT	Release complete message was received for this circuit's mate.
RSC_EVT	Reset message was received for this circuit's mate.
BLO_EVT	Blocking message was received for this circuit's mate.
BLA_EVT	Blocking acknowledgement message was received for this circuit's mate.
UBL_EVT	Unblocking message was received for this circuit's mate.
UBA_EVT	Unblocking acknowledgement message was received for this circuit's mate.
INF_EVT	Information message was received for this circuit's mate.
INR_EVT	Information request message was received for this circuit's mate.
CON_EVT	Connect message was received for this circuit's mate.
CPG_EVT	Call progress message was received for this circuit's mate.
SUS_EVT	Suspend message was received for this circuit's mate.
RES_EVT	Resume message was received for this circuit's mate.
SAM_EVT	Subsequent address message was received for this circuit's mate.
CGB_EVT	Circuit group blocking message was received concerning this circuit's mate.
CGU_EVT	Circuit group unblocking message was received concerning this circuit's mate.
ERR_EVT	Error indication was received for this circuit.

UDP to circuit

This event ID value passes information from the UDP thread to circuit threads:

Event	Description
CHKPNT_EVT	Checkpoint message was received for this circuit.

UDP to ISUP

This event ID value passes information from the UDP thread and ISUP thread:

Event	Description
APPREADY_EVT	Ready message was received from the mate application.

HMI to ISUP

The following event ID values pass information from the HMI thread to the ISUP thread:

Event	Description
DEAD_EVT	Halted, dead, or loading event was received from the Health Management service.
STARTED_EVT	Starting event was received from the Health Management service.
BACKUP_EVT	A now backup event was received from the Health Management service.
PRIMARY_EVT	A now primary event was received from the Health Management service.
STANDALONE_EVT	A now standalone event was received from the Health Management service.

isupdemo startup processes

The *isupdemo* startup processes are:

- Program startup
- Primary startup
- Backup startup

Program startup

The following table describes the program startup process for *isupdemo*:

Step	Action
1	At startup, the main thread parses the command line arguments, setting global variables based on the results of this parsing.
2	The user, test, HMI, UDP, and ISUP threads are started.
3	Incoming and outgoing threads are started.

Primary startup

The following table describes the primary startup process for *isupdemo*:

Step	Action
1	Upon receipt of the HMI_EVN_NOWPRIMARY event from the Health Management service, the HMI thread issues an EVT_PRIMARY event to the ISUP thread.
2	When the ISUP thread receives this event, it generates an application ready message to the mate application.
3	When an application ready message is received, the primary application initiates a batch checkpoint to the backup application.

Backup startup

The following table describes the backup startup process for *isupdemo*:

Step	Action
1	Upon receipt of the HMI_EVN_NOWBACKUP event from the Health Management service, the HMI thread issues an EVT_BACKUP event to the ISUP thread.
2	When the ISUP thread receives this event, it generates an application ready message to the mate application.
3	If an application ready message is received, the backup application again sends an application ready message to the primary application.

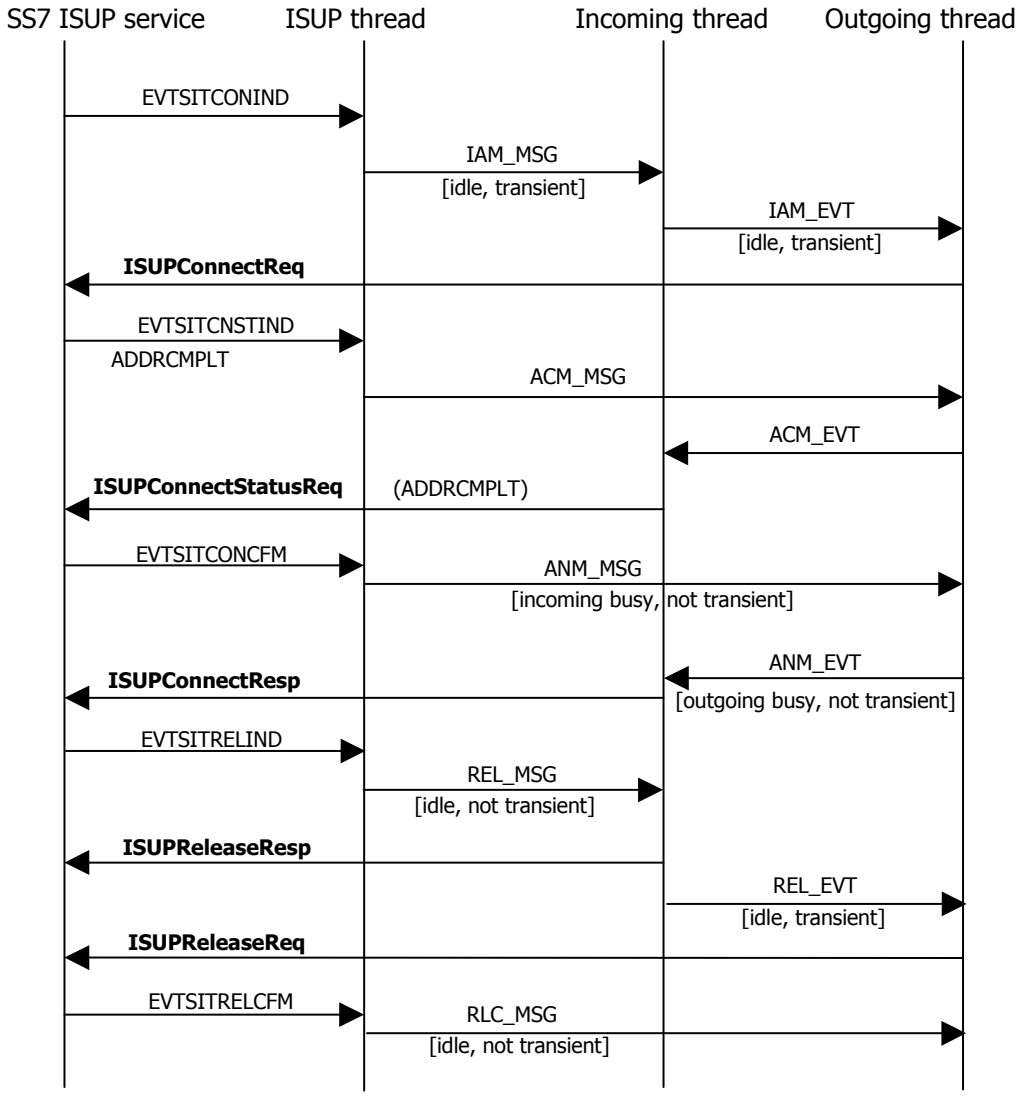
isupdemo call setup and release

When *isupdemo* is run, the following call setup and call release processes take place:

- Normal incoming call
- Incoming test call
- Outgoing test call

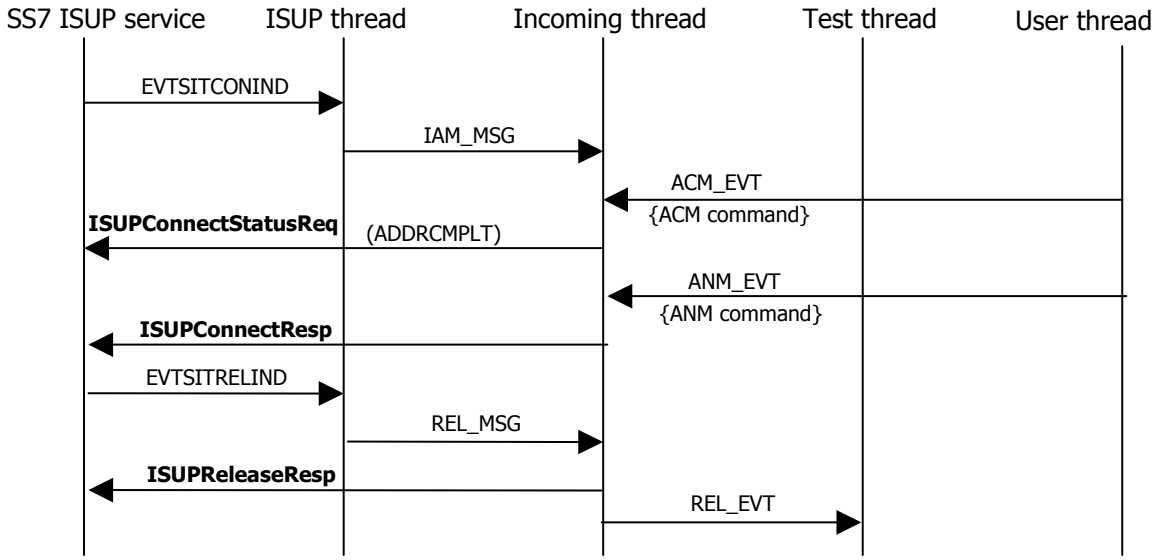
Normal incoming call

The following illustration shows a normal incoming call setup and release. Brackets ([]) indicate checkpoints, with the checkpointed data contained within the brackets in the form [***<circuit state>, <transient state>***].



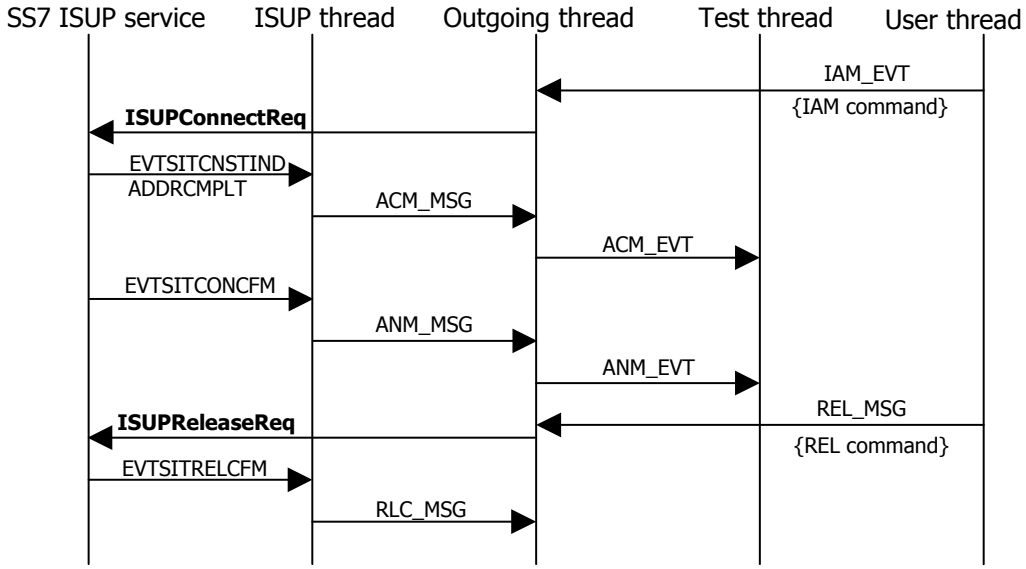
Incoming test call

The following illustration shows an incoming test call setup and release. Braces ({}) indicate commands entered from the keyboard.



Outgoing test call

The following illustration shows an outgoing test call setup and release. Braces ({}) indicate commands entered from the keyboard.



isupdemo command line options

isupdemo accepts the following command line options. Options can be entered in any order.

Option	Default	Description
-b ***boardNum***	1	Board to which this instance of the application communicates.
-da ***address***	Loopback (127.0.0.1)	Internet address of the mate application in dotted decimal format.
-dn ***name***	none	Name of the host on which the mate application is running.
-dp ***port***	4096	UDP port number of the mate application.
-lp ***port***	4096	UDP port for this application instance.
-n ***numCir***	64 (64 incoming circuits, 64 outgoing circuits)	Number of incoming and outgoing circuits.
-ni ***numIn***	64	Number of incoming circuits.
-no ***numOut***	64	Number of incoming circuits. **Note:** Incoming circuits are created first, starting with a circuit ID of one. Outgoing circuits are then created starting with circuit ID equal to the last incoming circuit ID plus one.
-s ***switchType***	ANSI92	ISUP switch type (ANSI88, ANSI92, ANSI95, ITUBLUE, ITUWHITE, Q767, or JNTT).
-te		Enables event tracing.
-tc		Enables checkpoint tracing.
-ta		Enables all tracing.

isupdemo user interface commands

Use the following commands to manage circuits and place and receive test calls. Test calls can only be received on incoming circuits and placed on outgoing circuits.

Command	Syntax	Description
QUIT	QUIT	Exits the application.
ACM	ACM ***circuit*** where ***circuit*** is the circuit ID of an incoming circuit.	Sends an address complete message.
ANM	ANM ***circuit*** where ***circuit*** is the circuit ID of an incoming circuit.	Sends an answer message.
BLO	BLO ***circuit*** where ***circuit*** is the circuit ID of the circuit to be blocked.	Sends a blocking message.
CGB	CGB ***circuit range*** where ***circuit*** is the circuit ID of the first circuit in the group to be blocked, and ***range*** is the desired range value.	Sends a circuit group blocking message.
CGU	CGU ***circuit range*** where ***circuit*** is the circuit ID of the first circuit in the group to be unblocked, and ***range*** is the desired range value.	Sends a circuit group unblocking message.
CON	CON ***circuit*** where ***circuit*** is the circuit ID of an incoming circuit.	Sends a connect message.
GRS	GRS ***circuit range*** where ***circuit*** is the circuit ID of the first circuit in the group to be reset, and ***range*** is the desired range value.	Sends a circuit group reset message.
IAM	IAM ***circuit called*** [***calling***] where ***circuit*** is the circuit ID of an outgoing circuit, ***called*** is the called party number, and ***calling*** is the optional calling party number.	Sends an initial address message.
REL	REL ***circuit*** [***cause***] where ***circuit*** is the circuit ID of the circuit to be released, and ***cause*** is an optional cause value.	Sends a release message.
RSC	RSC ***circuit*** where ***circuit*** is the circuit ID of the circuit to be reset.	Sends a circuit reset message.
UBL	UBL ***circuit*** where ***circuit*** is the circuit ID of the circuit to be unblocked.	Sends an unblocking message.

13 TCAP demonstration program

TCAP demonstration program overview

The TCAP demonstration program, *tcapdemo*, is a multiple-threaded program that uses the redundancy features of the SS7 TCAP layer. It is a skeletal implementation of a toll switch with a user interface for placing and receiving test calls.

tcapdemo data structures

The following table describes the data structures that are called by *tcapdemo:*

Structure	Description

ChkPntMsg	The checkpoint message structure transfers transaction information from the primary application to the backup application:
	`typedef struct checkPointMsg`
	`{`
	`  U32 msgId;  /* message Id */`
	`  U32 position;  /* position in "command.800" file */`
	`  U32 transId;  /* transaction Id */`
	`  U8 invokeId;  /* invoke Id */`
	`  U8 readFlag;  /* flaf for reading "command.800" file */`
	`} ChkPntMsg;`

Trans	The transaction structure passes information between threads by passing a pointer to an event in the buffer member of a CTA_EVENT structure:
	`typedef struct trans`
	`{`
	`  U32 position;`
	`  U32 transId;`
	`  U8 invokeId;`
	`  U8 readFlag;`
	`} Trans;`

tcapdemo threads

The following threads comprise *tcapdemo*:

Thread	Description
Main	Parses command line arguments, initializes global data, starts UDP thread, and sends or receives 800 number translation requests according to the transaction sequence defined in the *command.800* file. It can act as an 800 number server, or as a client requesting an 800 number translation.
UDP	Receives checkpoint messages from the mate application that result in checkpoint events generated to the main thread.

The commands.800 file

The transaction file, *commands.800*, informs the main thread of the transaction sequence to execute. This file has the following format:

Field	Description
qr_nl	Begin or query without permission message.
qr_l	Begin or query with permission message.
cv_nl	Continue or converse without permission message.
cv_l	Continue or converse with permission message.

UDP to TCAP event

This event ID value passes information from the UDP thread and main thread:

Event	Description
CHKPNT_EVT	A checkpoint message was received from the mate application.

tcapdemo command line options

The following command line options are accepted by *tcapdemo*. Options can be entered in any order. At the command line, enter the following command:

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

where ***options*** include:

Option	Default	Description
-b ***boardNum***	1	TX board number.
-p ***sapno***	0	Service access point ID. Valid range is 0 - 255.
-da ***address***	Loopback (127.0.0.1)	Internet address of mate application in dotted decimal format.
-dn ***name***		Host on which mate application is running.
-dp ***port***	4096	UDP port number of mate application.
-lp ***port***	4096	UDP port number for this application instance.
-n ***number***	254	Subsystem number to be used. Valid range is 0 - 255.
-i ***iterations***	1	Number of times transaction is repeated. Valid range is 0 - 32000.
-j ***delay***	1	Delay (in ms) between repetitions. Valid range is 1 - 65536.
-t		Uses ITU addressing. The *tcapdemo* program defaults to ANSI.
-s		Causes *tcapdemo* to act as an 800 number server. The *tcapdemo* program acts as a client by default.

The ***pointcode:subsystem*** parameter specifies the pointcode and subsystem number of the 800 number server. The ***phonenum*** parameter specifies the 800 number to be translated (only used by clients). Both parameters are used only by clients.

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

Acting as an 800 number server

Enter the following command to start *tcapdemo* as an 800 number server:

```
tcapdemo -b 1 -p 0 -n 255 -s
```

tcapdemo binds to TX board 1, uses SAP ID zero, and uses subsystem number 255. Since the `-s` parameter is specified, *tcapdemo* also acts as a server.

If binding is successful, *tcapdemo* receives a run state indication event from the TCAP task. *tcapdemo* uses the information it receives to determine its run status. The run status must be one of the following:

Run status	Description
Standalone or primary	*tcapdemo* waits for an 800 number request to arrive.
Backup	*tcapdemo* stops working as a server and waits for another run state indication event.

When a request arrives and run status is standalone or primary, *tcapdemo* compares the received 800 number to the information in the *numbers.800* file.

Note: The *numbers.800* file must be in the same directory as the *tcapdemo.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 conform to the following syntax:

```
800nnnnnnn=yyyyyyyyyy
```

If a matching 800 number is found, the *tcapdemo* server returns the translated number in a RETURN_RESULT [***last***] component.

If no matching 800 number is found, the *tcapdemo* server returns a RETURN_ERROR component.

The *tcapdemo* server continues to listen for and respond to requests indefinitely. To stop the server, press **Q**.

Acting as an 800 number client

Enter the following command to start *tcapdemo* as a client:

```
tcapdemo -b 2 -p 1 -j 100 -n 254 1.1.1:255 8001234567
```

In this case, *tcapdemo* binds to TX board 2, uses SAP ID one, and uses subsystem number 254. Because the -s parameter is not specified, *tcapdemo* acts as a client.

If binding is successful, *tcapdemo* receives a run state indication event from the TCAP task. *tcapdemo* uses the information it receives to determine its run status. The run status must be one of the following:

Run status	Description
Standalone or primary	*tcapdemo* sends an 800 number request to the specified pointcode and subsystem specified.
Backup	*tcapdemo* stops working as a client and waits for another run state indication event.

After sending the 800 number request, *tcapdemo* waits for a response.

After a response is received, *tcapdemo* continues to run, but no further requests are sent. To stop the client, press **Q**.

14 TUP demonstration program

TUP demonstration program overview

The TUP demonstration program, *tupdemo*, is a multiple-threaded program that uses the redundancy features of the SS7 TUP layer and the Health Management service. It is a skeletal implementation of a toll switch with a simple user interface for placing and receiving test calls and managing circuits.

tupdemo data structures

The following table describes the data structures*:*

Structure	Description

ChkPntMsg	The checkpoint message structure transfers circuit state information from the primary application to the backup application:
	`typedef struct checkPointMsg`
	`{`
	`  U32  msgId;`
	`  CirIdx cirId;  /* circuit ID of indicated circuit */`
	`  CirIdx mateId;  /* circuit ID of mate circuit  */`
	`  U16  state;  /* transient state indicator  */`
	`  U16  callState;  /* call processing state  */`
	`  U16  circuitState;  /* circuit blocking state  */`
	`} ChkPntMsg;`

Event	The event structure passes information between threads by passing a pointer to an event in the buffer member of a CTA_EVENT structure:
	`typedef struct`
	`{`
	`  TupRcvInfoBlk info;  /* TUP receive information block  */`
	`  SiAllSdus  sdu;  /* union of all TUP event structures  */`
	`  ChkPntMsg  chkPntMsg;  /* checkpoint message  */`
	`} Event;`

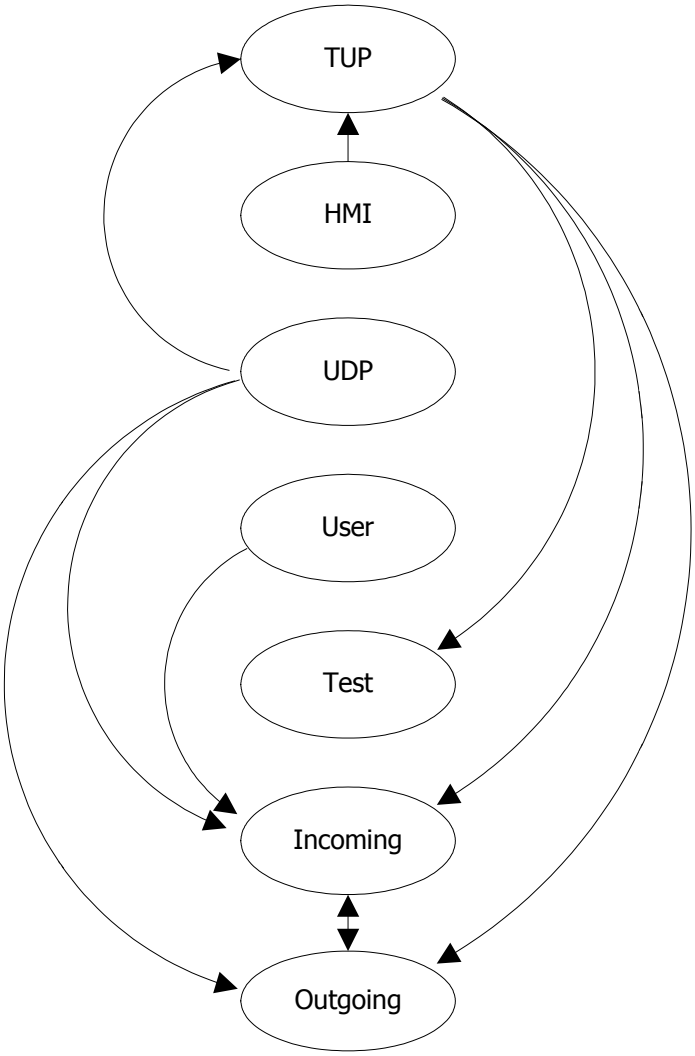
Circuit	The circuit control structure maintains information required by the application to control a particular circuit:
	`typedef struct circuit`
	`{`
	`  CTAQUEUEHD ctaQueue;  /* CTA queue for receiving events  */`
	`  CTAHD  ctaHndl;  /* CTA handle for this thread  */`
	`  CirIdx  cirId;  /* circuit ID of this circuit  */`
	`  CirIdx  mateId;  /* circuit ID for mate circuit  */`
	`  U16  state;  /* thread state  */`
	`  U16  callState;  /* call processing state  */`
	`  U16  circuitState;  /* circuit blocking state  */`
	`} Circuit;`

tupdemo threads

The following threads comprise *tupdemo*:

Thread	Description
Main	Parses command line arguments, initializes global data, and starts all other threads.
User	Accepts input from the keyboard and parses the input. Depending upon the command entered by the user, this results in an event to a circuit thread or a call to SS7 TUP.
Test	Receives events regarding a test call from a circuit thread. This results in a printed message indicating the type of event that was received.
UDP	Receives checkpoint messages from the mate application that result in checkpoint events generated to circuit threads. It also receives the application ready message from the mate application, resulting in an application ready event sent to the TUP thread.
HMI	Receives Natural Access events from the Health Management service and translates these events into internal events passed to the TUP thread.
TUP	Receives Natural Access events from SS7 TUP and translates these events into internal events routed to the appropriate incoming, outgoing, and test threads.
Incoming circuit	Receives events from the TUP thread and exchanges events with its mate outgoing thread.
Outgoing circuit	Receives events from the TUP thread and exchanges events with its mate incoming thread.

The following illustration shows the inter-thread communications in the TUP demonstration program:



tupdemo events

Threads use Natural Access to pass events between themselves. The topic describes the Natural Access event ID values used to identify these events.

- TUP to circuit
- Circuit to circuit
- UDP to circuit
- UDP to TUP
- HMI to TUP

TUP to circuit

The following event ID values pass information from the TUP thread to circuit threads. They also pass information from the user thread to circuit threads.

Event	Description
IAM_MSG	Initial address message was received for this circuit.
ACM_MSG	Address complete message was received for this circuit.
ANC_MSG	Answer message was received for this circuit.
CLF_MSG	Release (clear forward) message was received for this circuit.
RLG_MSG	Release guard message was received for this circuit.
CBK_MSG	Clear backward message was received for this circuit.
RSC_MSG	Reset message was received for this circuit.
BLO_MSG	Blocking message was received for this circuit.
BLA_MSG	Blocking acknowledgement message was received for this circuit.
UBL_MSG	Unblocking message was received for this circuit.
UBA_MSG	Unblocking acknowledgement message was received for this circuit.
RES_MSG	Resume message was received for this circuit.
MGB_MSG	Circuit group blocking message was received concerning this circuit.
MGU_MSG	Circuit group unblocking message was received concerning this circuit.
GRS_MSG	Circuit group reset message was received concerning this circuit.
IDLE_EVT	Circuit should immediately transition to the idle state with no TUP interaction.

Circuit to circuit

The following event ID values pass information between circuit threads:

Event	Description
IAM_EVT	Initial address message was received for this circuit's mate.
ACM_EVT	Address complete message was received for this circuit's mate.
ANC_EVT	Answer message was received for this circuit's mate.
CLF_EVT	Release (clear forward) message was received for this circuit's mate.
RLG_EVT	Release guard message was received for this circuit's mate.
CBK_EVT	Clear backward message was received for this circuit.
RSC_EVT	Reset message was received for this circuit's mate.
BLO_EVT	Blocking message was received for this circuit's mate.
BLA_EVT	Blocking acknowledgement message was received for this circuit's mate.
UBL_EVT	Unblocking message was received for this circuit's mate.
UBA_EVT	Unblocking acknowledgement message was received for this circuit's mate.
RES_EVT	Resume message was received for this circuit's mate.
MGB_EVT	Circuit group blocking message was received for this circuit's mate.
MGU_EVT	Circuit group unblocking message was received for this circuit's mate.
GRS_EVT	Circuit group reset message was received for this circuit.
ERR_EVT	Error indication was received for this circuit.

UDP to circuit

This event ID value passes information from the UDP thread to circuit threads:

Event	Description
CHKPNT_EVT	Checkpoint message was received for this circuit.

UDP to TUP

This event ID value passes information from the UDP thread and TUP thread:

Event	Description
APPREADY_EVT	A ready message was received from the mate application.

HMI to TUP

The following event ID values pass information from the HMI thread to the TUP thread:

Event	Description
DEAD_EVT	A halted, dead, or loading event was received from the Health Management service.
STARTED_EVT	A starting event was received from the Health Management service.
BACKUP_EVT	A now backup event was received from the Health Management service.
PRIMARY_EVT	A now primary event was received from the Health Management service.
STANDALONE_EVT	A now standalone event was received from the Health Management service.

tupdemo startup processes

The t*updemo* startup processes are:

- Program startup
- Primary startup
- Backup startup

Program startup

The following table describes the program startup process for *tupdemo*:

Step	Action
1	At startup, the main thread parses the command line arguments, setting global variables based on the results of this parsing.
2	The user, test, HMI, UDP, and TUP threads are started.
3	Incoming and outgoing threads are started.

Primary startup

The following table describes the primary startup process for *tupdemo*:

Step	Action
1	Upon receipt of the HMI_EVN_NOWPRIMARY event from the Health Management service, the HMI thread issues an EVT_PRIMARY event to the TUP thread.
2	When the TUP thread receives this event, it generates an application ready message to the mate application.
3	When an application ready message is received, the primary application initiates a batch checkpoint to the backup application.

Backup startup

The following table describes the backup startup process for *tupdemo*:

Step	Action
1	Upon receipt of the HMI_EVN_NOWBACKUP event from the Health Management service, the HMI thread issues an EVT_BACKUP event to the TUP thread.
2	When the TUP thread receives this event, it generates an application ready message to the mate application.
3	If an application ready message is received, the backup application again sends an application ready message to the primary application.

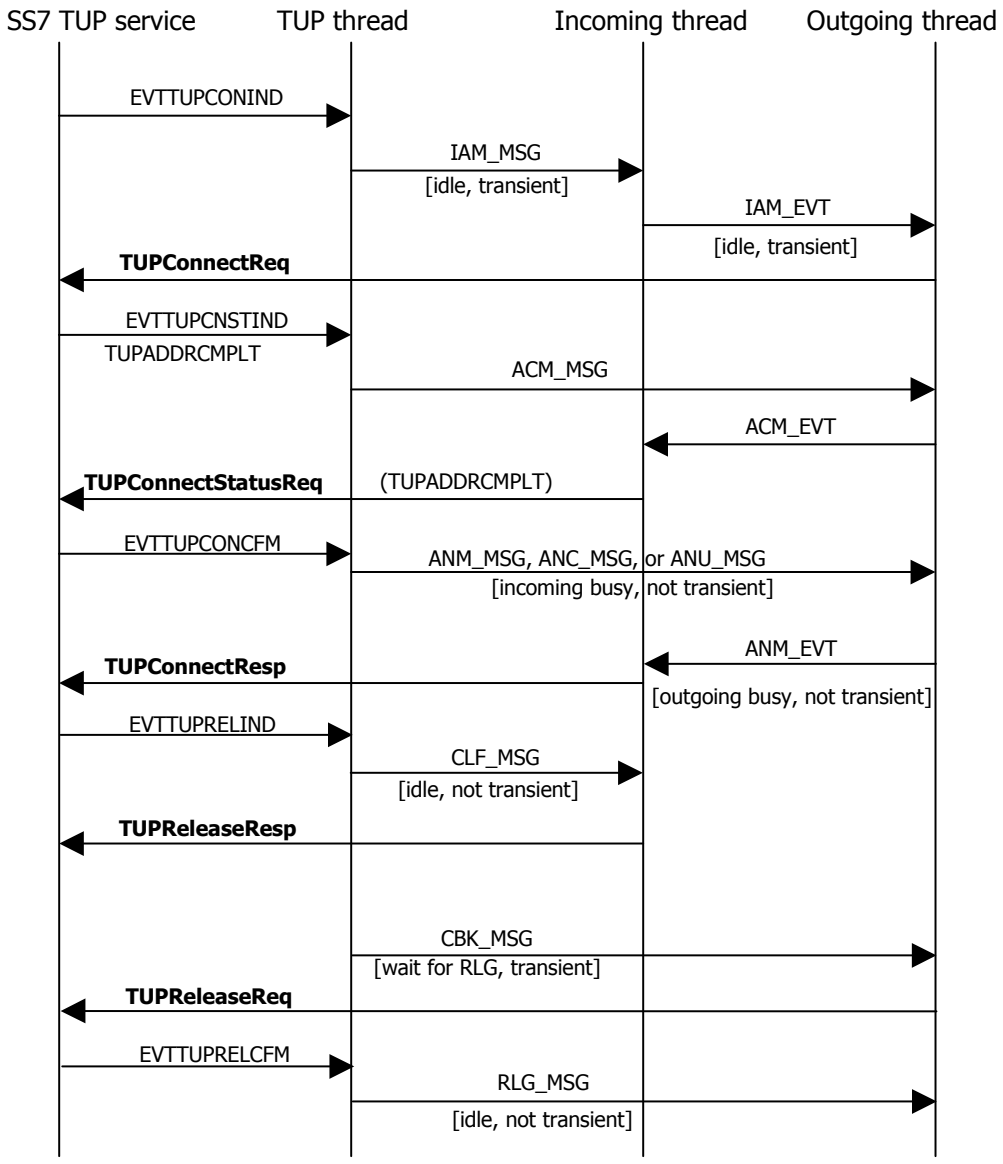
tupdemo call setup and release

When *tupdemo* is run, the following call setup and call release processes take place:

- Normal incoming call
- Incoming test call
- Outgoing test call

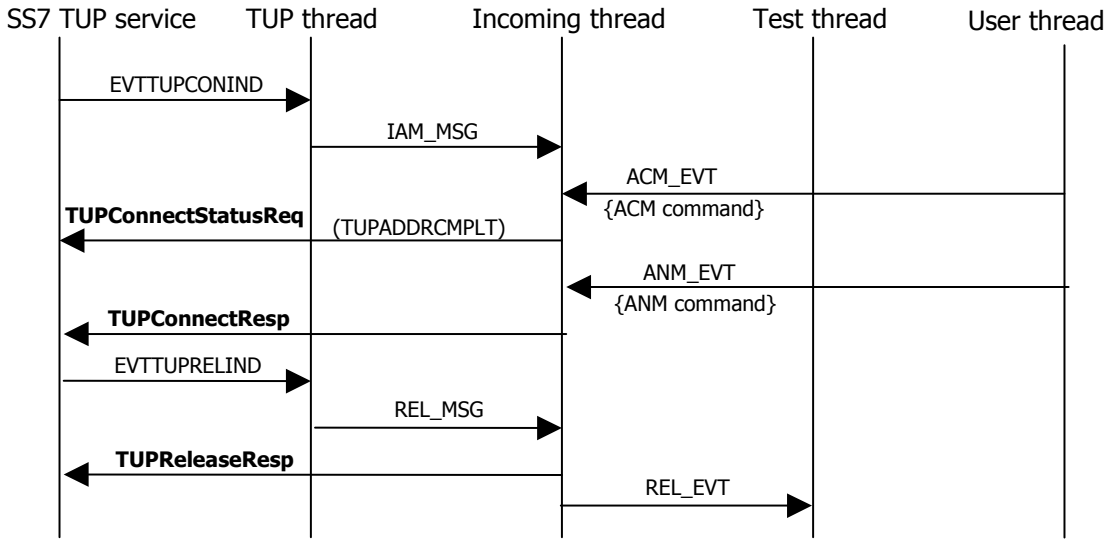
Normal incoming call

The following illustration shows a normal incoming call setup and release. Brackets ([]) indicate checkpoints, with the checkpointed data contained within the brackets in the form [***<circuit state>, <transient state>***].



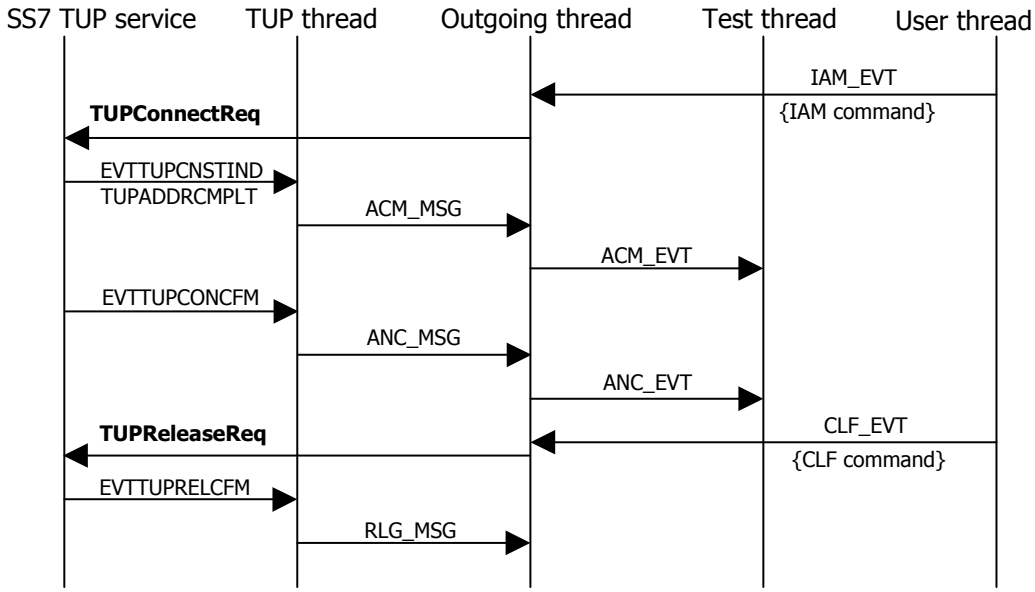
Incoming test call

The following illustration shows an incoming test call setup and release. Braces ({}) indicate commands entered from the keyboard.



Outgoing test call

The following illustration shows an outgoing test call setup and release. Braces ({}) indicate commands entered from the keyboard.



tupdemo command line options

The following command line options are accepted by the *tupdemo* demonstration program. Options can be entered in any order.

Option	Default	Description
-b ***boardNum***	1	Board to which this instance of the application communicates.
-da ***address***	Loopback (127.0.0.1)	Internet address of the mate application in dotted decimal format.
-dn ***name***	none	Name of the host on which the mate application is running.
-dp ***port***	4096	UDP port number of the mate application.
-lp ***port***	4096	UDP port for this application instance.
-n ***numCir***	12 (12 incoming circuits, 12 outgoing circuits)	Number of incoming and outgoing circuits.
-ni ***numIn***	12	Number of incoming circuits.
-no ***numOut***	12	Number of incoming circuits. **Note:** Incoming circuits are created first, starting with a circuit ID of one. Outgoing circuits are then created starting with circuit ID equal to the last incoming circuit ID plus one.
-s ***switchType***	ITU	TUP switch type (ITU, CTW).
-te		Enables event tracing.
-tc		Enables checkpoint tracing.
`-ta`		Enables all tracing.

tupdemo user interface commands

Use the following commands to manage circuits and place and receive test calls. Test calls can only be received on incoming circuits and placed on outgoing circuits.

Command	Syntax	Description
QUIT	QUIT	Exits the application.
ACM	ACM ***circuit*** where ***circuit*** is the circuit ID of an incoming circuit.	Sends an address complete message.
ANC	ANC ***circuit*** where ***circuit*** is the circuit ID of an incoming circuit.	Sends an answer message.
BLO	BLO ***circuit*** where ***circuit*** is the circuit ID of the circuit to be blocked.	Sends a blocking message.
MGB	MGB ***circuit range*** where ***circuit*** is the circuit ID of the first circuit in the group to be blocked, and ***range*** is the desired range value	Sends a circuit group blocking message.
MGU	MGU ***circuit range*** where ***circuit*** is the circuit ID of the first circuit in the group to be unblocked, and ***range*** is the desired range value.	Sends a circuit group unblocking message.
GRS	GRS ***circuit range*** where ***circuit*** is the circuit ID of the first circuit in the group to be reset, and ***range*** is the desired range value.	Sends a circuit group reset message.
IAM	IAM ***circuit called*** [***calling***] where ***circuit*** is the circuit ID of an outgoing circuit, ***called*** is the called party number, and ***calling*** is the optional calling party number.	Sends an initial address message.
CLF	CLF ***circuit*** [***cause***] where ***circuit*** is the circuit ID of the circuit to be released, and ***cause*** is an optional cause value.	Sends a CLF (clear forward) message.
RSC	RSC ***circuit*** where ***circuit*** is the circuit ID of the circuit to be reset.	Sends a circuit reset message.
UBL	UBL ***circuit*** where ***circuit*** is the circuit ID of the circuit to be unblocked.	Sends an unblocking message.

Index

U