Dialogic®

Making Innovation Thrive™

Dialogic® NaturalAccess™ TUP Layer Developer's Reference Manual

July 2009 64-0463-01

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.

Any use case(s) shown and/or described herein represent one or more examples of the various ways, scenarios or environments in which Dialogic® products can be used. Such use case(s) are non-limiting and do not represent recommendations of Dialogic as to whether or how to use Dialogic products.

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.

The 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.

Revision history

Revision	Release date	Notes
9000-6473-20	January 1998	B.2.0 release
9000-6473-21	July 1998	GJG
9000-6473-22	September 1998	GJG
9000-6473-23	March 1999	GJG
9000-6473-24	January 2000	GJG
9000-6473-25	July 2000	GJG, SS7 3.5 feature pack
9000-6473-27	June 2001	GJG, SS7 3.8 beta
9000-6473-28	November 2003	SRG, SS7 4.0
9000-6473-29	April 2005	SRG, SS7 4.2
9000-6473-31	July 2006	LBZ, SS7 4.3
9000-6473-31	September 2008	LBG, SS7 5.0
64-0463-01	July 2009	LBG, SS7 5.1
Last modified: June 23, 2009		

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

Table Of Contents

1 Introduction

The *Dialogic® NaturalAccess™ TUP Layer Developer's Reference Manual* explains how to implement the SS7 TUP (telephone user part) layer using NaturalAccess™ TUP. This manual explains how to create applications using NaturalAccess™ TUP and presents a detailed specification of its signaling procedures and functions.

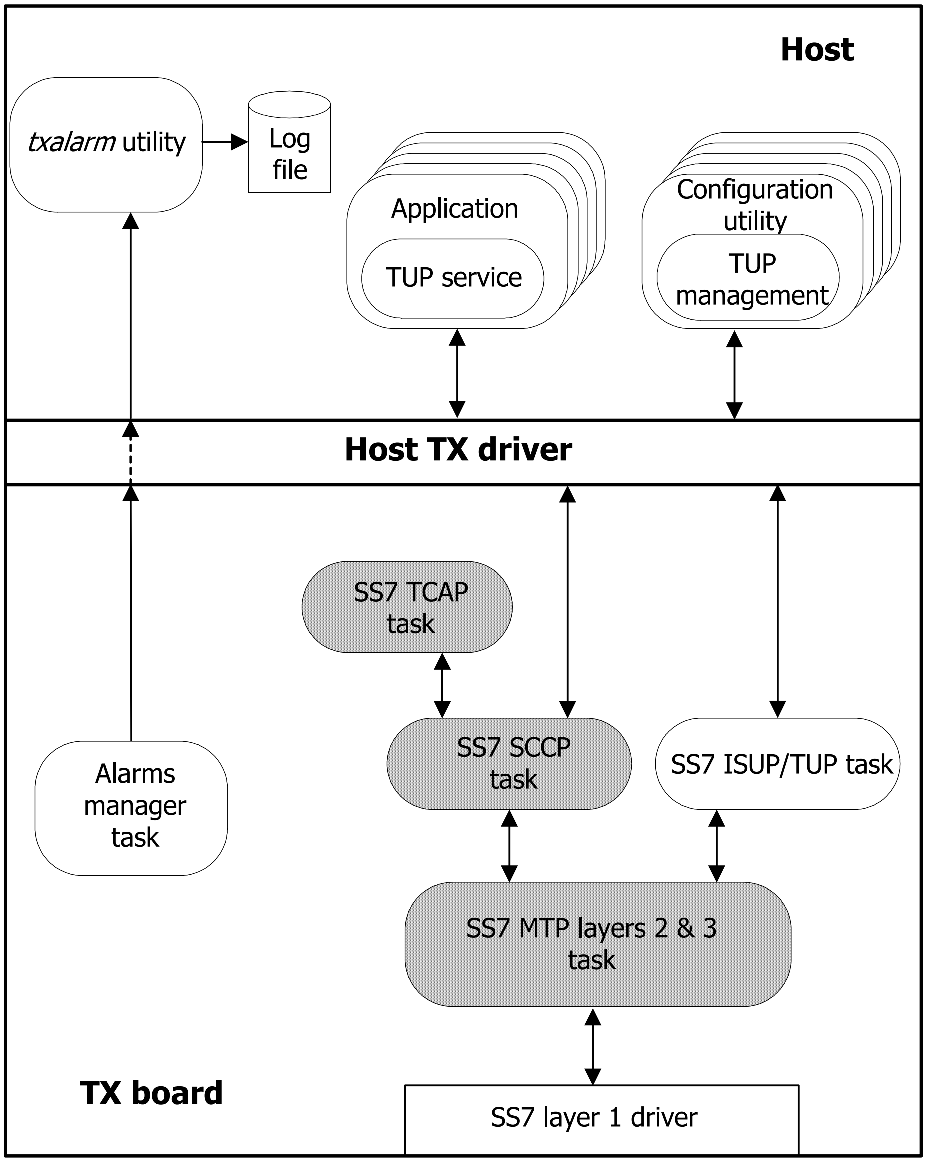
Note: The product(s) to which this document pertains is/are among those sold by NMS Communications Corporation ("NMS") to Dialogic Corporation ("Dialogic") in December 2008. Certain terminology relating to the product(s) has been changed, whereas other terminology has been retained for consistency and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

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

2 SS7 overview

SS7 architecture

The following illustration shows the SS7 architecture in a typical system with separate host applications handling the data and control (TUP) interface, system configuration, and system alarms:



The TX board consists of the following components:

- TUP task that implements the SS7 TUP layer.
- TUP configuration text file that describes the circuit groups and routing instructions employed by the TUP layer in establishing, supervising, and maintaining circuit switched connections.
- MTP task that implements the SS7 MTP 2 (data link) layer and the SS7 MTP 3 (network) layer.
- Optional SCCP task that implements the SS7 SCCP layer.
- Optional ISUP task that implements the SS7 ISUP layer.
- Optional TCAP task that implements the SS7 TCAP layer.
- The TX alarms manager task that collects unsolicited alarms (status changes) generated by the SS7 tasks and forwards them to the host for application-specific alarm processing.

The host consists of the following components:

- A TX driver for the native host operating system that provides low-level access to the TX board from the host.
- Functions that provide the application with a high-level interface to the TUP layer services (*tupapi.lib* and *tupapi.dll*).
- Functions that provide the application with a high level interface for task and circuit management and configuration (*tupmgmt.lib* and *tupmgmt.dll*).
- An alarm collector process for capturing alarms and saving them to a text file. The alarm collector (*txalarm*) is provided in both executable and source form. The source can be used as an example for developers who want to integrate the TX alarms into their own alarm monitoring system.
- Configuration utilities (one for each SS7 layer) that read the SS7 configuration file(s) and load the configurations to the TX processor tasks at system startup. The TUP configuration utility (*tupcfg*) is provided in both executable and source form. The source code can be used as an example for developers who want to integrate the TUP configuration into their own configuration management system.
- Demonstration programs showing the use of the TUP service. The *tuporig* demonstration program makes an outgoing phone call. The *tupterm* demonstration program accepts incoming phone calls.
- The TUP manager utility (*tupmgr*) provides a command line interface from which alarm levels can be set, buffers can be traced, and TUP statistics can be viewed and reset.

TUP task

The SS7 TUP task provides the interface for applications to establish, maintain, and clear circuit switched connections through the SS7 network in accordance with the ITU-T (CCITT) recommendations Q.721-Q.725 and GF 001-9001 (China). TUP is not used in ANSI networks, so there is no applicable ANSI standard for TUP.

The TUP task maintains a database of circuits and circuit groups that are controlled by the application and keeps track of the state of each circuit. The initial characteristics of each circuit (group), such as the circuit identification code (CIC) and destination point code are specified in the TUP configuration file. The TUP task reads the TUP configuration file at startup time.

For outgoing call setup requests, the application specifies the circuit to be connected. For incoming calls, the TUP task verifies that the circuit state and characteristics are compatible with the incoming call request parameters before passing the incoming call indication to the application. For both incoming and outgoing calls, the TUP task provides all necessary connection timers, notifying both the application and the far exchange with necessary indications such as connection clearing when critical timers expire.

The TUP task:

- Provides circuit supervision for the duration of the connection.
- Adjusts the circuit state as needed based on requests from the application and TUP messages received from the far exchange.
- Provides connect and disconnect timing.
- Handles circuit (group) blocking and unblocking, updating the state of the affected circuits as needed.
- Detects protocol errors on behalf of the application.

3 TUP programming model

Programming model overview

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

TUP service users

NMS TUP supports applications with service access points, or SAPs. One user service access point is defined for each application that uses the TUP service. At initialization, applications bind to a particular user SAP by specifying the SAP ID. Each user service access point in the TUP configuration file is associated with a switch type (ITU-T or CHINA) that the TUP task uses to associate with its configured network connections. A single user SAP, and hence a single application, for a particular switch type can be defined for outgoing call requests. If an application supports multiple TX boards, it must bind with each board separately.

The TUP task must bind with the MTP task through the use of network service access points (NSAP). The MTP configuration must define the network service access points (only ITU-T switch type) with which TUP can bind to communicate with the MTP task. The TUP configuration file must specify the MTP network service access point to which to bind to communicate with the MTP task. An ANSI TUP does not exist.

Note: The characteristics of each service access point are specified in the TUP configuration file. Refer to the *NMS SS7 Configuration Manual* for more information.

Entity and instance IDs

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

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

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

NMS TUP functions

NMS TUP provides two sets of functions:

- Service functions
- Management functions

Service functions

The TUP service functions provide the application access to the TUP layer services. Applications invoke TUP services by calling TUP request functions that send a TUP message to a remote exchange or endpoint. Request function parameters are converted to messages for processing by the TUP task.

The TUP requests from the remote endpoints are presented to the application as indications, using the same driver and mechanisms through which confirmations are delivered. The application then issues a reply to the endpoint by invoking the appropriate TUP service response function.

All TUP service functions are asynchronous. Completion of the function implies only that the function was successfully initiated (a request message was queued to the TUP task). Errors detected by the TUP task result in asynchronous status indications being sent to the application. Successfully delivered requests generally result in no notification to the application until the far end takes some corresponding action such as returning a connect confirm message in response to a connection request.

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

Applications detect that an event is pending through an operating system specific mechanism such as **poll** in UNIX or **WaitForMultipleObjects** in Windows. The application retrieves the event data (or message) through a function that also translates the confirmation parameters from SS7 TUP raw format to API format.

For more information, refer to the *Using the TUP service function reference* on page 43.

Management functions

The TUP management functions manage the following TUP tasks on the board:

- Configuration
- Circuits
- Statistics and status retrieval

For more information, refer to *TUP management overview* on page 59 and the TUP management function reference section.

Queues and contexts

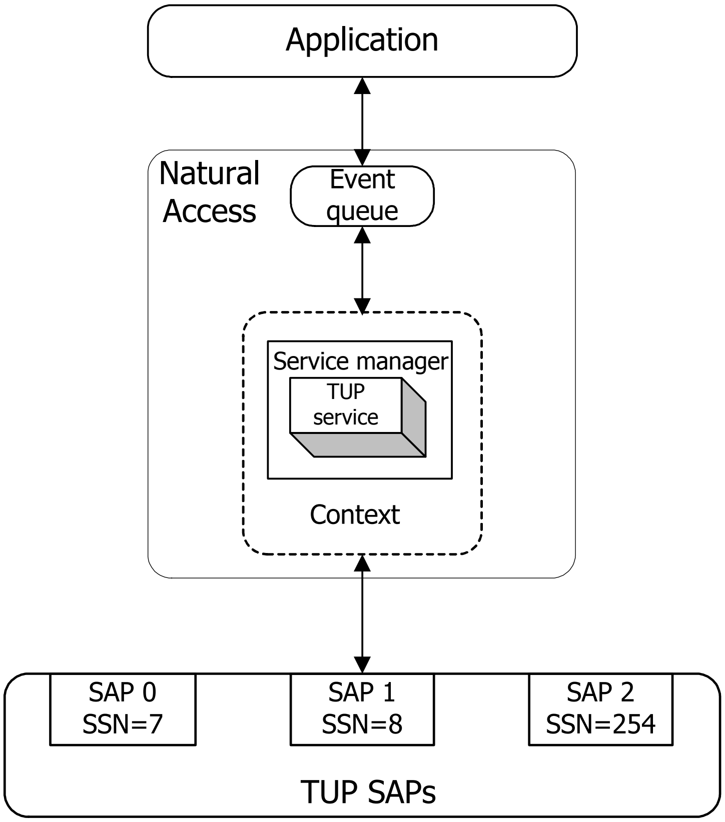
Natural Access organizes services and their associated resources around a processing object known as a context. Each instance of an application binding to a TUP service access point is a unique Natural Access context. Contexts are created with **ctaCreateContext**.

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

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

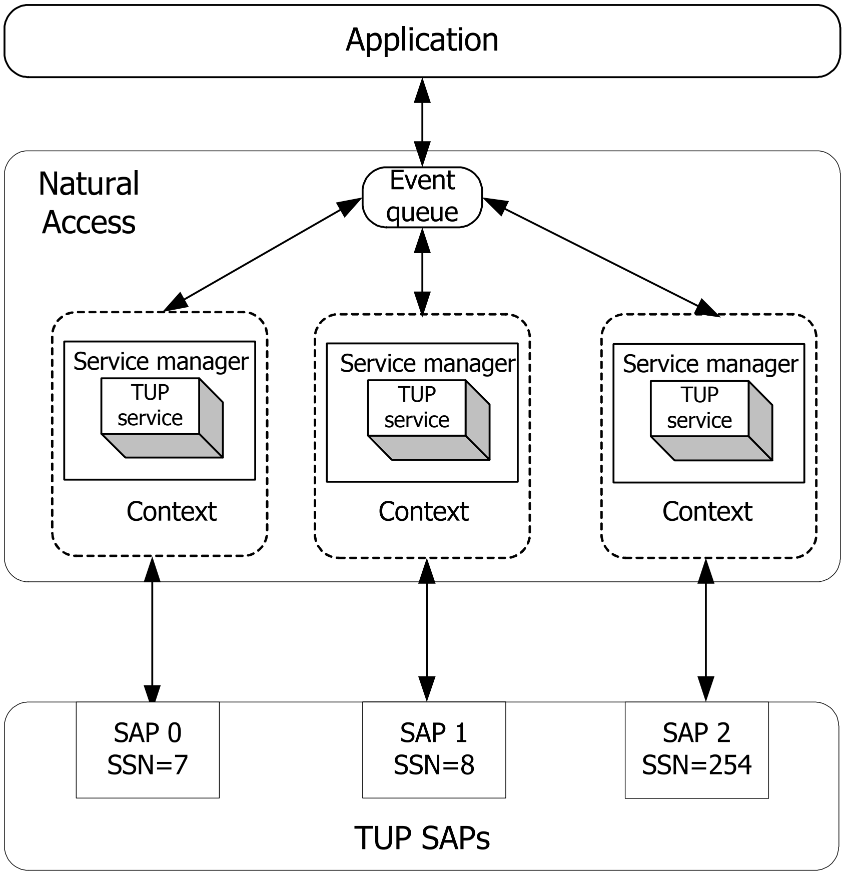
Single-context, single-queue model

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



Multiple-context, single-queue model

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

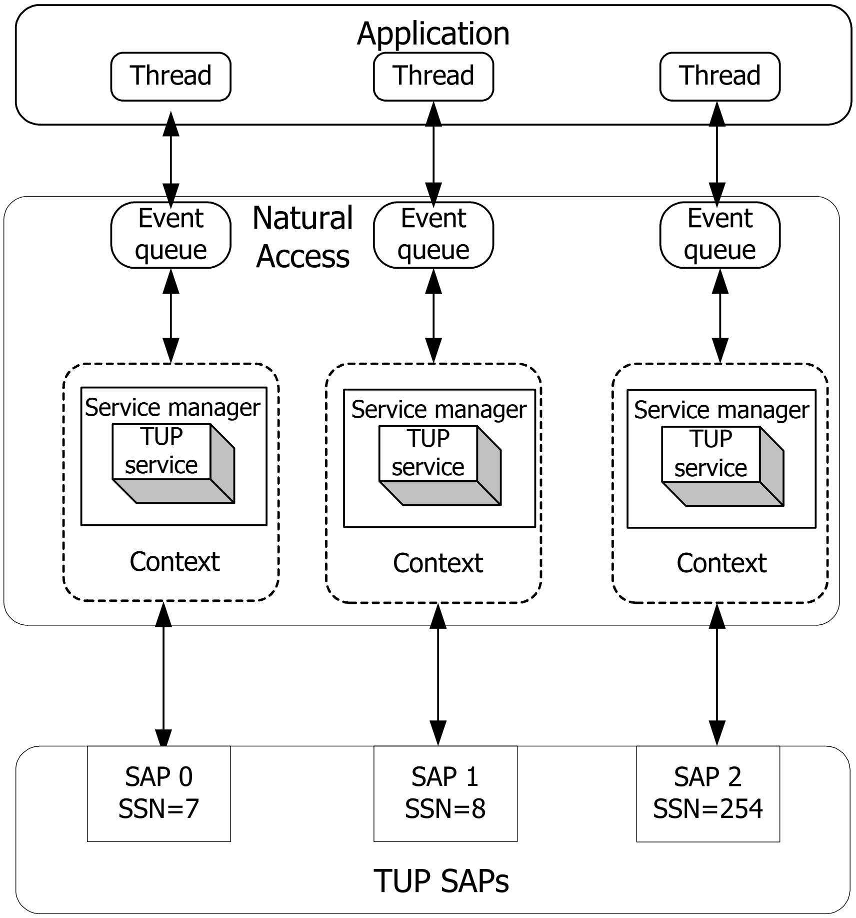


Multiple-context, multiple-queue model

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

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

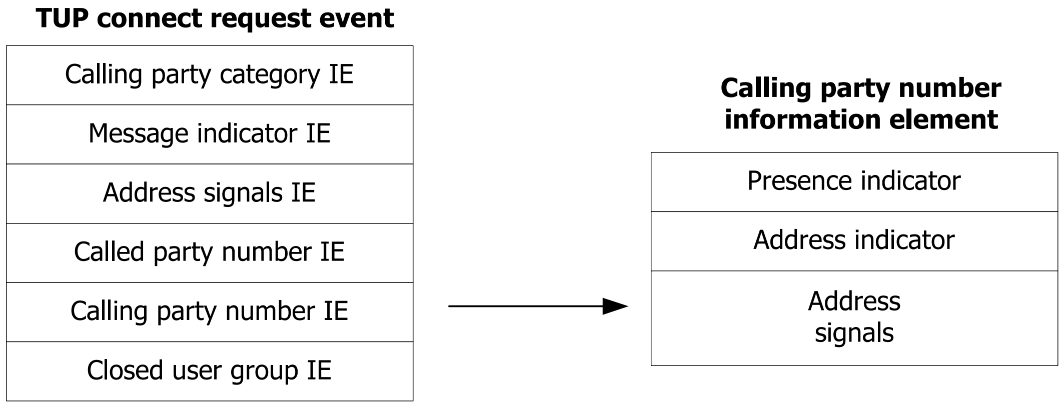
The following illustration shows a multiple-context, multiple-queue programming model:



Signaling parameters

Signaling parameters are passed between the application and the TUP task in the form of events. Events are fixed format structures consisting of one or more information elements (IEs). Information elements are fixed format structures consisting of a flag indicating their presence or absence from the corresponding TUP message, and one or more tokens, or fields.

The following illustration shows how event structures simplify applications by enabling them to operate on fixed format structures rather than the variable length and variable formats employed by the TUP protocol:



Operating system specific considerations

Applications receive confirmations (responses) and indications of far-end initiated requests by periodically calling **TUPRetrieveMessage**. This function checks for an incoming message and returns an indication of whether or not a message is available. When an incoming message is received, **TUPRetrieveMessage**:

- Copies the event structure to the buffer provided by the caller
- Performs any byte-order translation between network byte order and host byte order
- Reports which event was received back to the calling application

The application must periodically call this function to receive messages from the TUP layer.

To allow applications more flexibility in handling multiple devices, mechanisms are provided to allow the calling application to be notified when a TUP message is available for it without having to poll the TX board by periodically calling **TUPRetrieveMessage**.

Multiple-threaded considerations

In a multiple-threaded application, any thread can generate a request. However, a single receiver thread must receive all asynchronous events from the TUP layer (incoming calls, status indications, and so on) and route them to the proper transaction thread based on the circuit, service user instance ID, or other transaction information.

For example, in an application consisting of a main thread that spawns a child call thread whenever it wants to initiate a call, the call thread is responsible for initiating **TUPConnectReq**, collecting the responses from the receiver thread, and taking the appropriate action. The application can have many simultaneous calls and call threads active at any given time.

Multiple call threads can generate the transaction requests in any order. The requests are processed by the TUP layer in the order they are received. If each call thread then calls **TUPRetrieveMessage** to retrieve the response to its transaction request, there is no guarantee that the TUP message returned by **TUPRetrieveMessage** would be associated with the calling thread's call. The first thread to call **TUPRetrieveMessage** when a message is pending receives the first pending message, regardless of the transaction with which it is associated. The incoming message can be a new call request from a far service switching point or a network status indication message.

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

Unsolicited alarms

Any of the tasks on the TX processor (including any of the SS7 layers) can generate unsolicited alarms at any time to indicate abnormal events or changes in the status of an entity. The *txalarm* utility is provided to receive alarms and write them to a disk file on the host.

The following table displays the alarm text and the alarm number. The headings are the code from the *txalarm* utility for these fields (ap is an alarm pointer).

Note: Inbound and outbound calls reuse certain alarm numbers. Any code looking for these alarms must check the call state of the circuit, as well as the string of the alarm.

If these messages are generated by a timer popping, it is noted in parentheses after the alarm number.

ap->data.infalmstr[s1len+1]	**ap->data.infalarm**
"Unknown/unreasonable packet received [cirIdx]"	3864
"Circuit Block remains in effect [cirIdx]"	3861 (Timer 11)
"Circuit Block ack not received [cirIdx]"	3862 (Timer 13)
"Circuit unblock ack not received [cirIdx]"	3863 (Timer 16)
"Ceasing transmission of call failures [cirIdx]"	3879 (Timer 5)
"Continuity Checks Failing [cirIdx]"	3874
"Connection Machine In reset circuit [cirIdx]"	3875
"Release Guard Not Received [cirIdx]"	3877 (Timer 7)
"Waiting for Continuity Check Result [cirIdx]"	3874 (Timer 1)
"Waiting for Continuity Re-Check Request [cirIdx]"	3875 (Timer 9 or 10)
"Connection Machine Out reset circuit [cirIdx]"	3876
"HGB request unacknowledged [dpc:cic:range]"	4009 (Timer 33)
"HGU request unacknowledged [dpc:cic:range]"	4010 (Timer 35)
"Remote MGB still in effect [dpc:cic:range]"	4007 (Timer 25)
"MGB request unacknowledged [dpc:cic:range]"	4005 (Timer 27)
"Local MGB still in effect [dpc:cic:range]"	4007 (Timer 25)
"MGU request unacknowledged [dpc:cic:range]"	4006 (Timer 29)
"GRS request unacknowledged [dpc:cic:range]"	4008 (Timer 22)
"SGB request unacknowledged [dpc:cic:range]"	4011 (Timer 39)
"SGU request unacknowledged [dpc:cic:range]"	4012 (Timer 41)
"Message received for unequipped circuit [CIC]"	3938
"Connection Machine In could not get spInstId [cirIdx]"	3876
"Connection Machine In could not get Sdu [cirIdx]"	3878
"Connection Machine started Continuity Check In[cirIdx]"	3877
"Connection Machine Out could not get spInstId [cirIdx]"	3879
"Connection Machine Out started continuity check [cirIdx]"	3878
"Second Continuity Check Failure [cirIdx]"	3888
"BACKUP: Data indications and Data requests rejected"	4063

4 Using the TUP service

Setting up the Natural Access environment

Before calling any TUP service functions, the application must:

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

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

Initializing the Natural Access environment

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

```
CTA_INIT_PARMS      tupInitparms       = {0};
CTA_SERVICE_NAME    tupServiceNames[]  = {{"TUP", "TUPMGR"}};
...
tupInitparms.size              = sizeof(CTA_INIT_PARMS);
tupInitparms.traceflags        = CTA_TRACE_ENABLE;
tupInitparms.parmflags         = CTA_PARM_MGMT_SHARED;
tupInitparms.ctacompatlevel    = CTA_COMPATLEVEL;

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

Creating queues and contexts

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

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

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

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

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

Binding to the TUP service

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

Field	Description
board	TX board number.
srcEnt	Calling application entity ID.
srcInst	Calling application instance ID.
suId	Calling application service user ID.
spId	TUP service access point ID on which to bind.
ssn	TUP subsystem number associated with the service access point.

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

```
CTA_SERVICE_DESC tupOpenSvcLst[] = {{{"TUP", "TUPMGR"}, {0}, {0}, {0}}};

tupOpenSvcLst[0].svcargs.args[0] = board;   /* board number           */
tupOpenSvcLst[0].svcargs.args[1] = INST_ID; /* srcInst                */
tupOpenSvcLst[0].svcargs.args[2] = ENT_ID;  /* srcEnt                 */
tupOpenSvcLst[0].svcargs.args[3] = 1;       /* AutoBind? (yes=1,no= 0)*/
tupOpenSvcLst[0].svcargs.args[4] = SAP_ID;  /* spId                   */
tupOpenSvcLst[0].svcargs.args[5] = SAP_ID;  /* suId                   */
tupOpenSvcLst[0].svcargs.args[6] = poolsize /* poolsize               */
```

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

If multiple contexts are assigned to the same queue, all of the contexts must use the same entity ID in the service arguments parameter. Conversely, contexts bound to different queues must specify a unique entity ID.

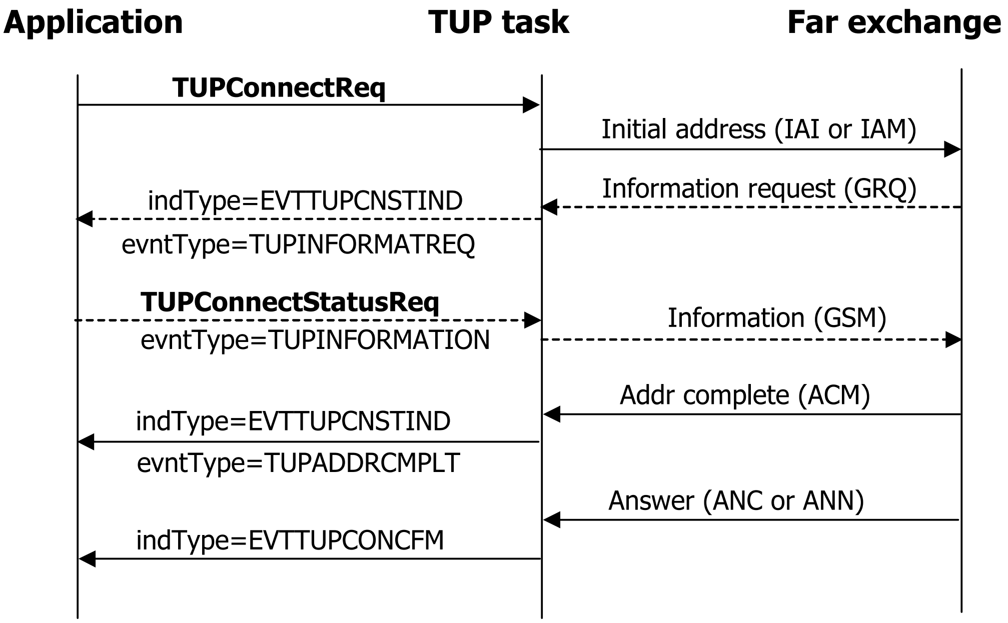
```
CTA_EVENT   event;    /* Event structure to wait for TUP events */
...


Ret = ctaOpenServices( ctaHd, tupOpenSvcLst, 1 );
if ( Ret != SUCCESS )
{
    ctaGetText( NULL_CTAHD, Ret, sErr, sizeof( sErr ) );
    printf( "ERROR : ctaOpenServices failed( %s )\n", sErr );
    ctaDestroyQueue( ctaQueue );  /* destroys context too */
    return(...)
}

/* Wait for "open services" to complete; note: this loop
 * assumes no other contexts are already active on the queue
 * we're waiting on, so no other events will be received that
 * need handling
 */
event.id = CTAEVN_NULL_EVENT;
do
{
    ctaWaitEvent( ctaQueue, &event, 5000 );
}
while( (event.id != CTAEVN_OPEN_SERVICES_DONE) &&
       (event.id != CTAEVN_WAIT_TIMEOUT) );

/* check if binding succeeded */
if( (pSap->event.id != CTAEVN_OPEN_SERVICES_DONE) ||
    (pSap->event.value != CTA_REASON_FINISHED) )
{
    ctaGetText( event.ctahd, event.value, sErr, sizeof( sErr ) );
    printf( "ERROR opening TUP service [%s]\n", sErr );
    ctaDestroyQueue( pSap->ctaQueue );    /* destroys context too */
    return( ... );
}
```

Establishing connections

This topic describes how to establish outgoing and incoming connections.

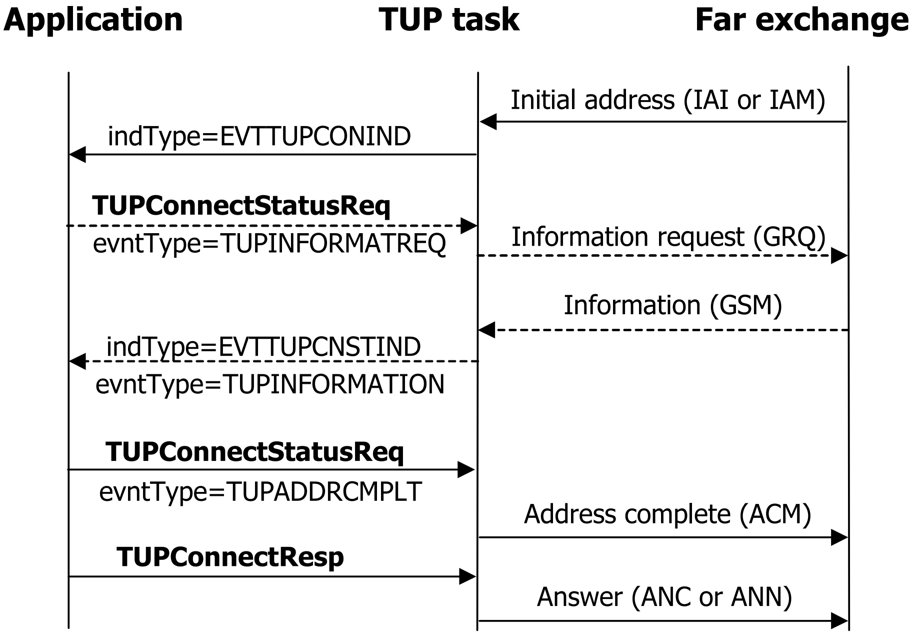
Establishing outgoing connections

The application initiates a circuit switched connection by invoking **TUPConnectReq** resulting in the generation of a TUP initial address message (IAM or IAI) to the far exchange. The far exchange can initiate the connection by sending the IAM or IAI message. The application receives a TUP connect indication (EVTTUPCONIND) event. The following illustration shows the process for establishing outgoing connections (where dashed lines indicate optional sequences):



Establishing incoming connections

During the connection establishment phase, the application exchanges call progress and other status information with the far exchange by invoking **TUPConnectStatusReq** with an event type and by receiving TUP connect status indication (EVTTUPCNSTIND) events from the TUP task. The following illustration shows the process for establishing incoming connections (where dashed lines indicate optional sequences):



The connection establishment phase ends when the application:

- Receives the TUP connect confirmation (EVTTUPCONCFM) event (far exchange sent answer or connect message).
- Invokes **TUPConnectResp** to signal to the far end that the connection is established for an incoming call.

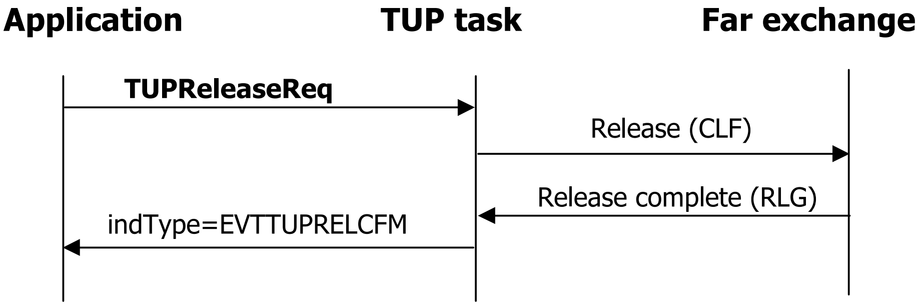
Clearing connections

This topic describes how connections are cleared under the following circumstances:

- Application clearing outgoing connection
- Far end exchange clearing outgoing connection
- Application clearing incoming connection
- Far end exchange clearing incoming connection

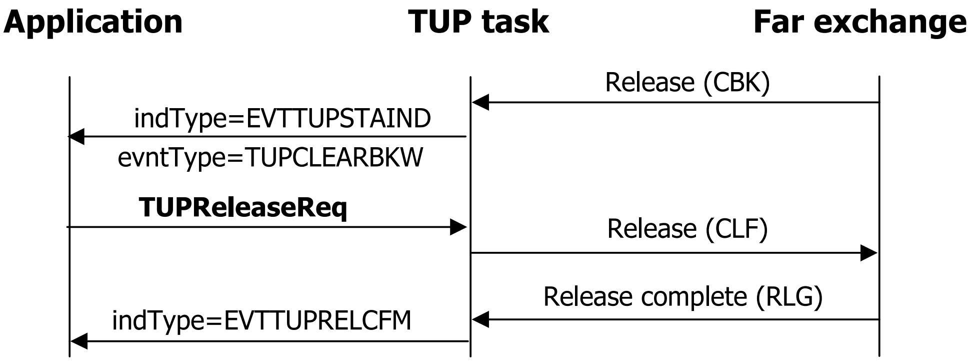
Application clearing far end connection

The application requests clearing the outgoing connection by invoking **TUPReleaseReq**. The application is notified of the completion of the release procedure (the receipt of a release complete message) when it receives a TUP release confirm (EVTTUPRELCFM) event. The following illustration shows the process by which the application performs outgoing connection release:



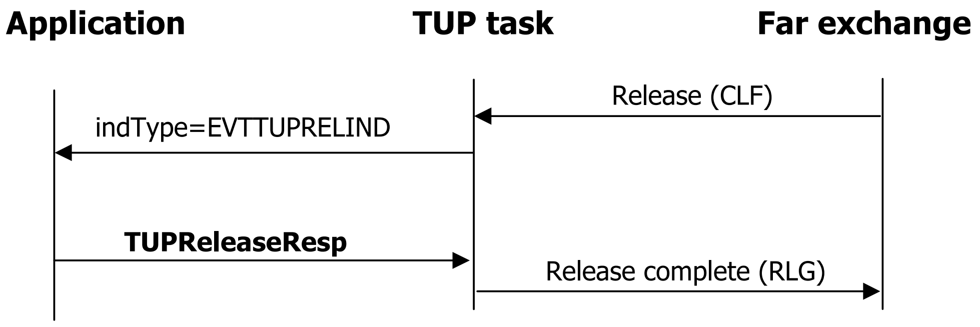
Far end exchange clearing outgoing connection

If the far exchange initiates the release of the outgoing connection, the application receives a TUP release indication event from the TUP layer. The application then completes the connection release by invoking **TUPReleaseReq** to send the release message to the far exchange and wait for the TUP release confirm (EVTTUPRELCFM) event. The following illustration shows the process by which far exchange connection release for outgoing connections occurs:



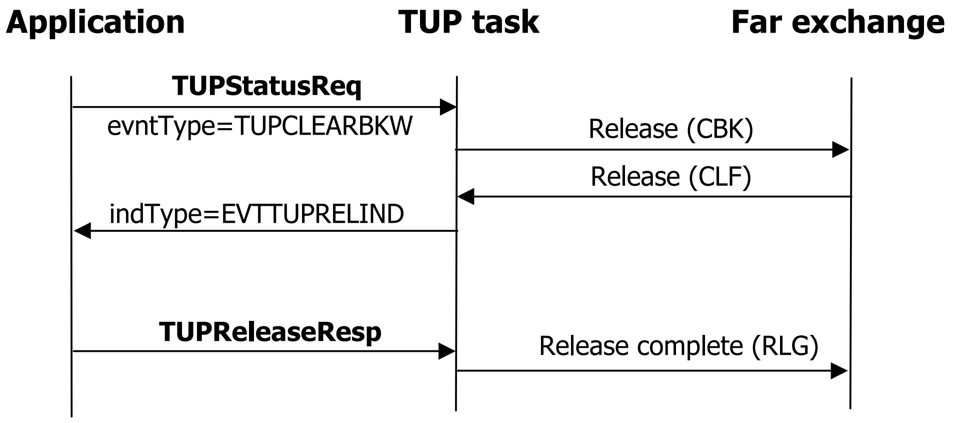
Application clearing incoming connection

If the far exchange initiates the release of its own outgoing connection, the application receives a TUP release indication (EVTTUPRELIND) event from its incoming connection of the TUP layer. The following illustration shows the process by which the application performs far exchange connection release for incoming connections:



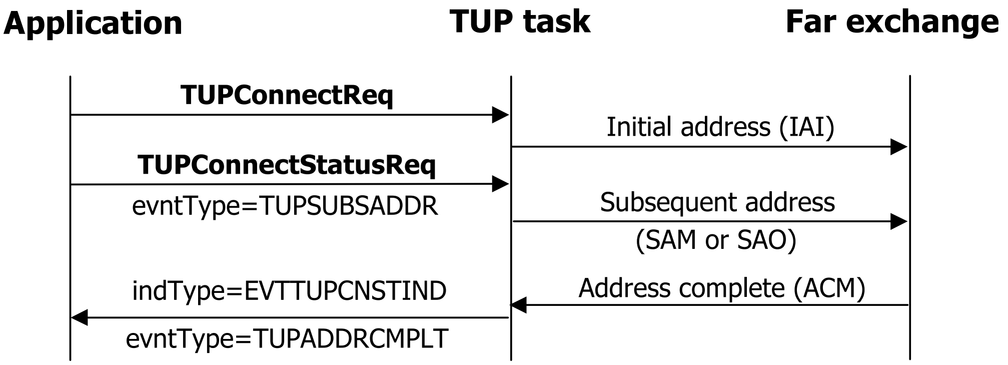
Far end exchange clearing incoming connection

To clear an incoming connection, the application starts with **TUPStatusReq** and waits for TUPReleaseInd to return. The application can then answer with **TUPReleaseResp**. The following illustration shows the process by which the application performs incoming connection release:



Sending subsequent address messages

After calling **TUPConnectReq**, the application can send subsequent address messages (SAM or SAO) with **TUPConnectStatusReq**. The far exchange generates an address complete message when the address is complete. The application must wait for a TUPStatusInd after sending the last SAM or SAO. The following illustration shows the process by which the application performs subsequent address connection establishment:



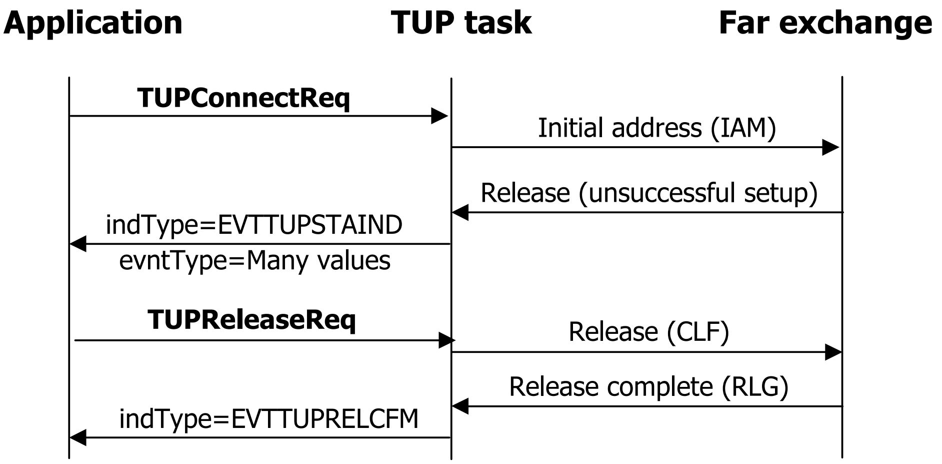
Handling failed connection requests

This topic shows are events are generated and how cleanup occurs when **TUPConnectReq** connection requests fail in the following situations:

- Circuit blocked
- Circuit reset
- Subscriber busy
- Unknown or missing response

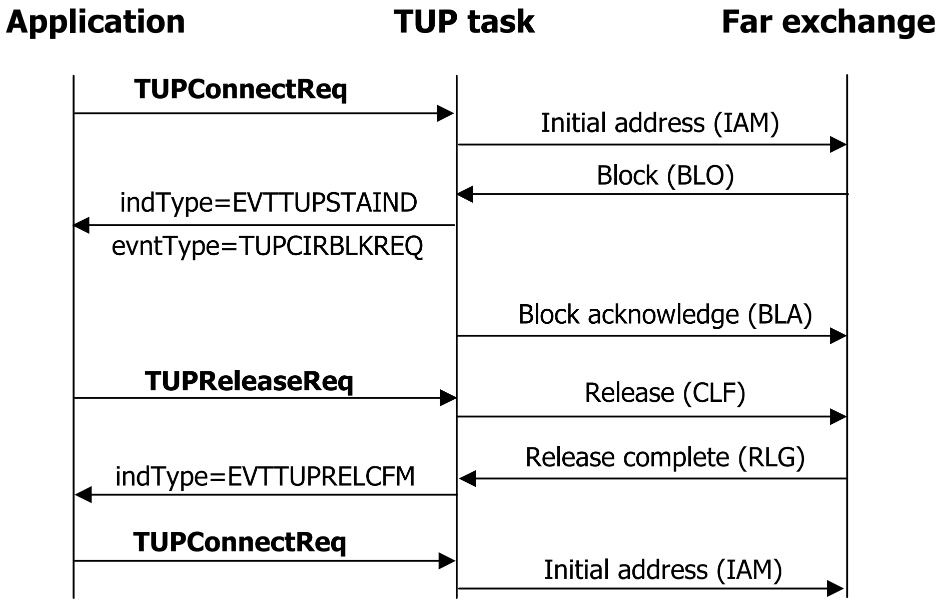
Subscriber busy

If the called number is busy, **TUPConnectReq** can fail. The following illustration shows how events are generated and how cleanup occurs (before a reconnection is attempted) when a connection fails because the subscriber is busy:



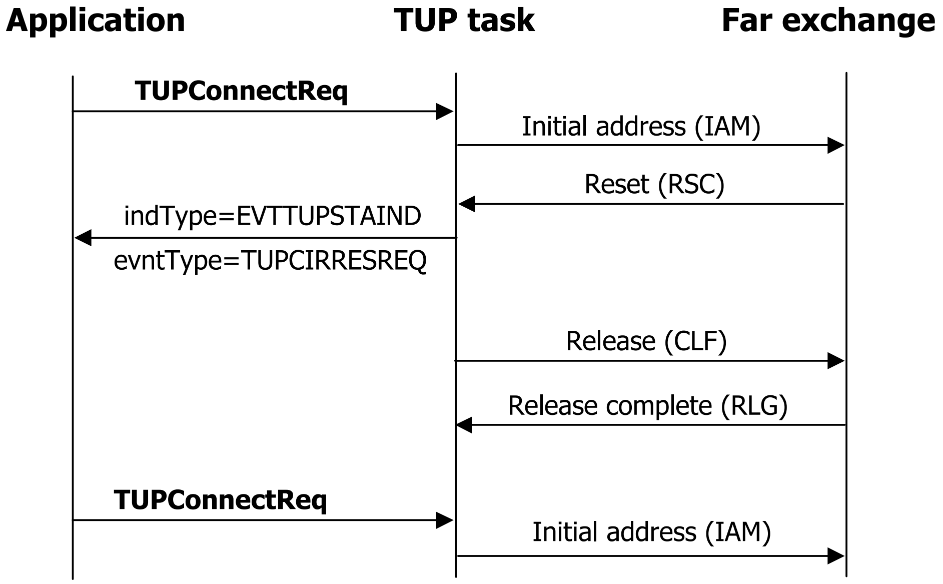
Circuit blocked

If the far exchange blocked the circuit, **TUPConnectReq** can fail. The following illustration shows how events are generated and how cleanup occurs (before a reconnection is attempted) when a connection fails because the circuit is blocked:



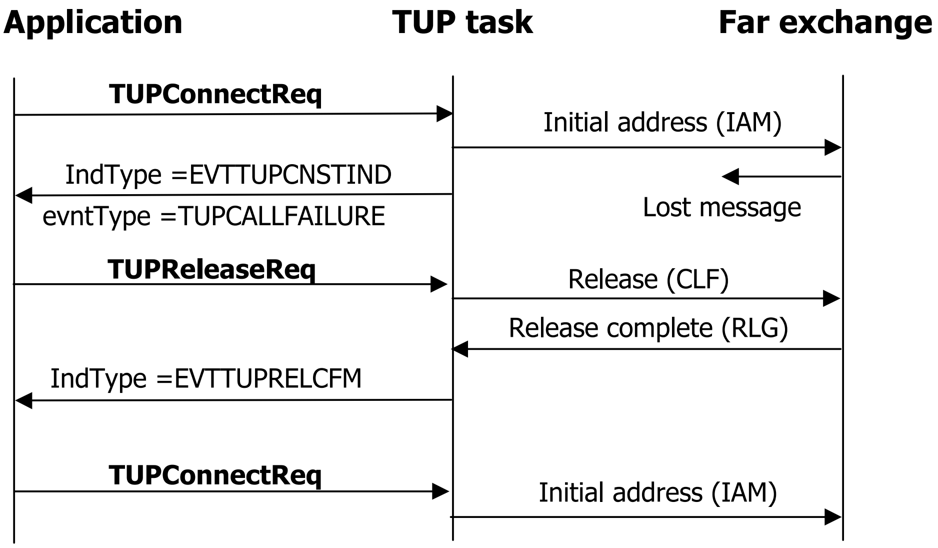
Circuit reset

If the far end exchange resets the circuit, **TUPConnectReq** can fail. The following illustration shows how events are generated and how cleanup occurs (before a reconnection is attempted) when a connection fails because the circuit has been reset:



Unknown or missing response

If an intelligible answer to the initial address message (IAM) is not received, **TUPConnectReq** can fail. The following illustration shows how events are generated and how cleanup occurs (before a reconnection is attempted) when a connection fails because there is an unknown or missing response:

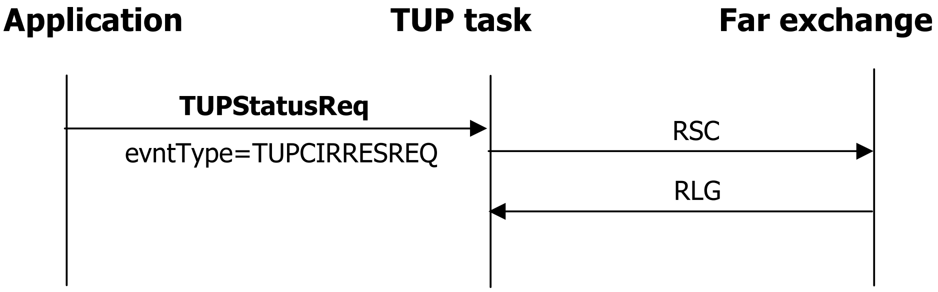


Resetting circuits

This topic describes the circuit reset process when initiated by the application and the far exchange.

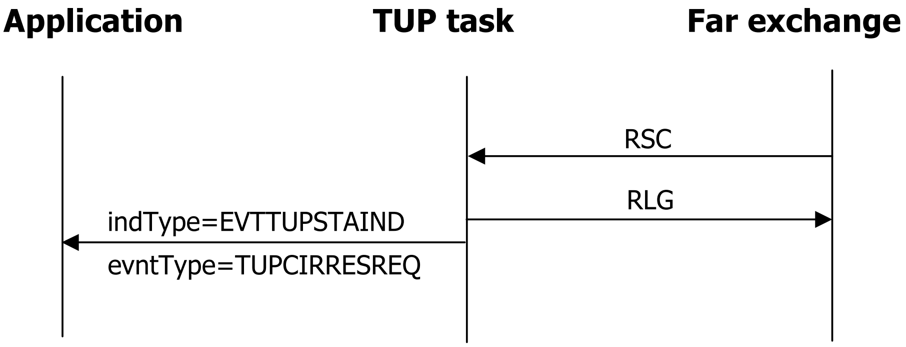
Application initiated circuit reset

The application requests the reset of a circuit by invoking **TUPStatusReq** with the event type of TUPCIRRESREQ. The application can consider the circuit reset upon issuing this request. The following illustration shows the application-initiated circuit reset process:



Far exchange initiated circuit reset

If the far exchange initiates the reset of the circuit, the application receives a status indication (EVTTUPSTAIND) with the event type of TUPCIRRESREQ from the TUP layer. The application can consider the circuit reset upon receiving this indication. The TUP task acknowledges the reset request by sending a release complete message (RLG). The following shows the far exchange-initiated circuit reset process:

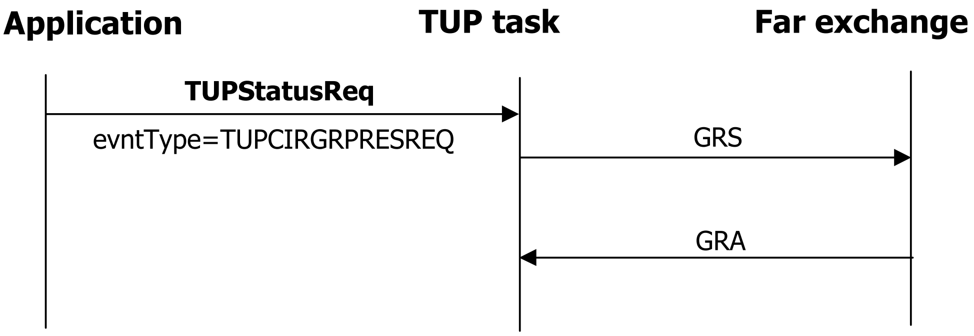


Resetting circuit groups

This topic describes the circuit group reset process when performed by the application and the far exchange.

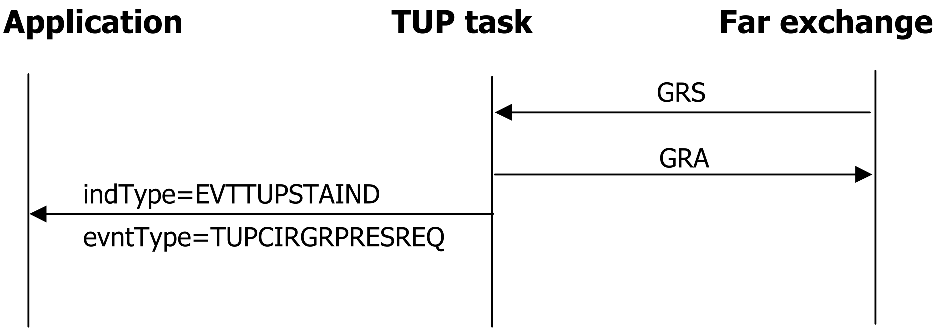
Application initiated circuit reset

The application requests the reset of a circuit group by invoking **TUPStatusReq** with the event type of TUPCIRGRPRESREQ. In the call to **TUPStatusReq**, the application must specify the starting circuit number and the range of circuits within a group. The application can consider the circuit group reset upon issuing this request. The following illustration shows how the application initiates a group rest request:



Far exchange initiated circuit reset

If the far exchange initiates the reset of the circuit group, the application receives a status indication (EVTTUPSTAIND) with the event type of TUPCIRGRPRESREQ from the TUP layer for the circuit group. The application can consider these circuits reset upon receiving this indication. The TUP task acknowledges the group reset request by sending a group reset acknowledgment message (GRA). The following illustration shows the process by which a far exchange-initiated group reset request is conducted:



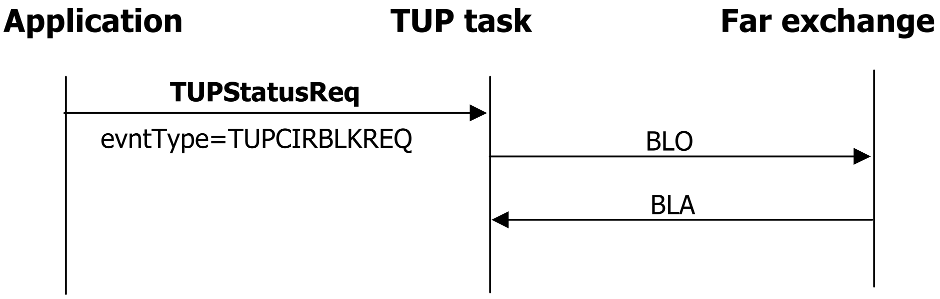
Blocking and unblocking circuits

This topic describes the following processes for blocking and unblocking circuits:

- Application blocking a circuit
- Far exchange blocking a circuit
- Application unblocking a circuit
- Far exchange unblocking a circuit

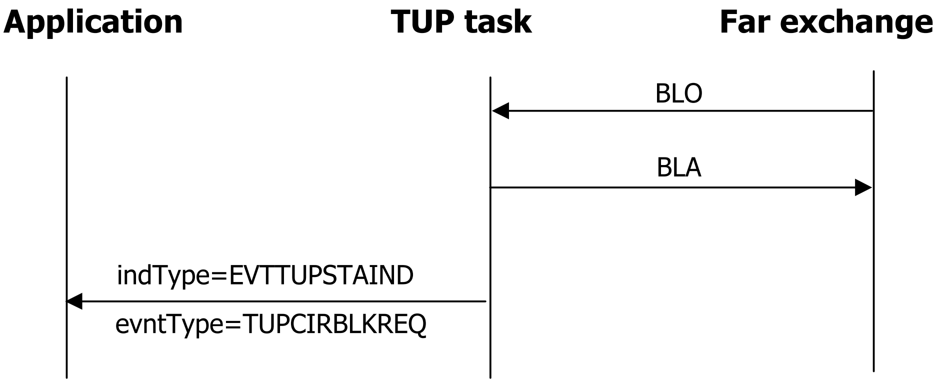
Application blocking a circuit

The application requests blocking of a circuit by invoking **TUPStatusReq** with the event type of TUPCIRBLKREQ. The application can consider the circuit blocked or unblocked when issuing this request. The following illustration shows how the application initiates a blocking request:



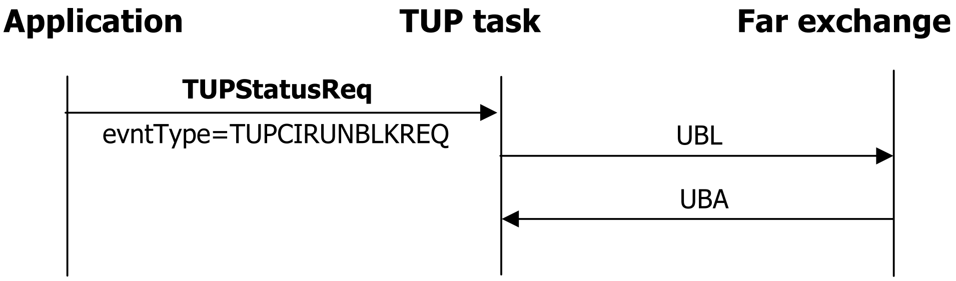
Far exchange blocking a circuit

If the far exchange initiates the blocking of the circuit, the application receives a status indication (EVTTUPSTAIND) with the event type of TUPCIRBLKREQ from the TUP layer. The TUP task acknowledges the block for the application. The following illustration shows how a far exchange-initiated blocking request is conducted:



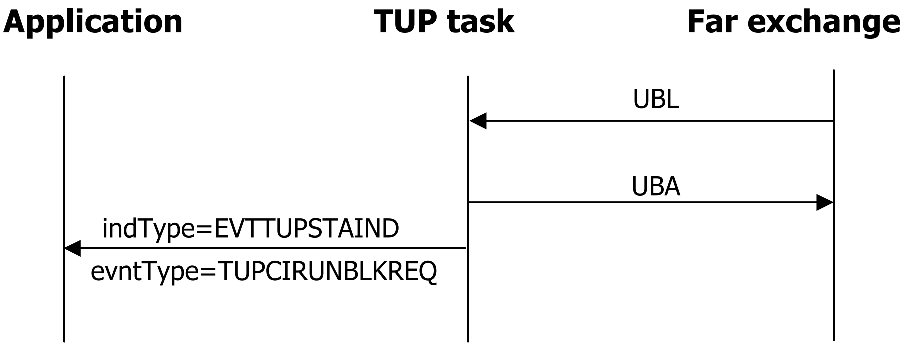
Application unblocking a circuit

The application requests unblocking of a circuit by invoking **TUPStatusReq** with the event type of TUPCIRUNBLKREQ. The following illustration shows how the application initiates an unblocking request:



Far exchange unblocking a circuit

If the far exchange initiates the unblocking of the circuit, the application receives a status indication (EVTTUPSTAIND) with the event type of TUPCIRUNBLKREQ from the TUP layer. The TUP task then acknowledges the circuit unblocking. The following illustration shows how a far exchange-initiated unblocking request is conducted:



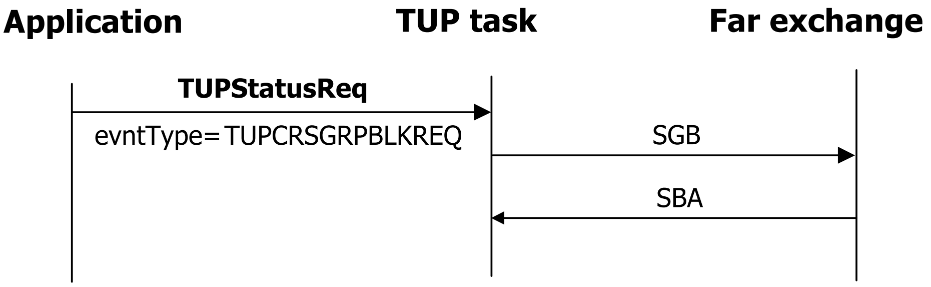
Blocking and unblocking software-oriented circuit groups

This topic describes the following processes for blocking and unblocking circuits:

- Application initiated group blocking request
- Far exchange initiated group blocking request
- Application initiated group unblocking request
- Far exchange initiated group unblocking request

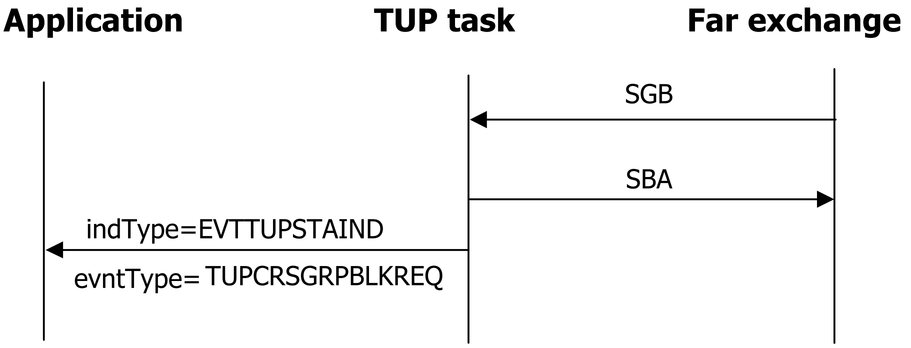
Application-initiated group blocking request

The application requests blocking of a circuit group by invoking **TUPStatusReq** with the event type of TUPCRSGRPBLKREQ. The application can consider the circuit group blocked or unblocked when issuing this request. The following illustration shows how the application initiates a group blocking request:



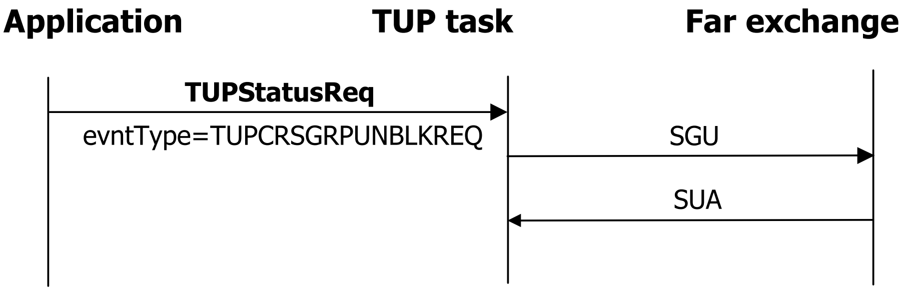
Far exchange-initiated group blocking request

If the far exchange initiates the blocking of the circuit group, the application receives a status indication (EVTTUPSTAIND) with the event type of TUPCRSGRPBLKREQ from the TUP layer. The TUP task acknowledges the circuit group blocking. The following illustration shows how a far exchange-initiated group blocking request is conducted:



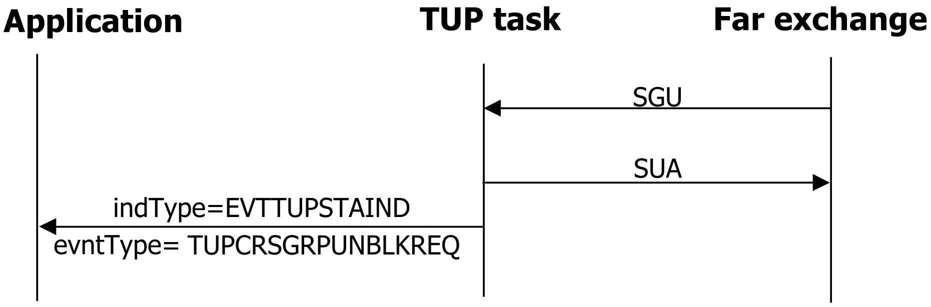
Application initiated group unblocking request

The application requests unblocking of a circuit group by invoking **TUPStatusReq** with the event type of TUPCRSGRPUNBLKREQ. The following illustration shows how the application initiates a group unblocking request:



Far exchange initiated group unblocking request

If the far exchange initiates the unblocking of the circuit group, the application receives a status indication (EVTTUPSTAIND) with the event type of TUPCRSGRPUNBLKREQ from the TUP layer. The TUP task then acknowledges the circuit group unblocking. The following illustration shows how a far exchange-initiated group unblocking request is conducted:



Controlling TUP congestion

The TUPEVN_CONGESTION event indicates one of the following congestion issues:

- Memory usage on the TX board has become very high.
- The queue in the TUP service is growing.

In either case, you get a congestion level of 0 through 3 in the value element of the CTA_EVENT structure.

If your application receives a level 1 event, reduce the number of calls being generated. At levels 2 and 3 , avoid all new calls and clear existing calls. As memory usage lowers or the outbound queue shrinks, congestion events with lower congestion levels are generated for the application to resume more normal traffic.

If your application requires additional information about the congestion, call **TUPGetApiStats** for additional statistics.

Setting up tracing

TUP allows tracing of all packets passed between the TUP task and the MTP task. To enable tracing, follow these steps:

Step	Action
1	Enable packet tracing by starting the TUP manager program (*tupmgr*) and entering the following command at the prompt: `TRACE DATA ON`
2	Start the application.
3	In a separate window, enter the following command: `ss7trace -f` ***`filename`*** where ***filename*** is the trace log file to be opened. Packets are dumped to the screen in the *ss7trace* window.

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

Handling redundancy events

After binding to a TUP user SAP, the application receives a status indication indicating the MTP redundancy or run state on the board. The event type associated with status indication (EVTSITSTAIND) indicates one of the following states:

State	Description
TUPMTP3STANDALONE	Application is in a non-redundant configuration. Normal operation can begin.
TUPMTP3PRIMARY	The run state of MTP is primary on this board in a redundant board pair. Normal operation is allowed as long as the board remains primary.
TUPMTP3BACKUP	The run state of MTP is backup on this board in a redundant board pair, monitoring the status of the primary board. No active traffic passes through this SAP until the board becomes the primary member of the pair.
TUPMTP3RESUME	MTP resume indication is delivered to the application when the TUP layer receives a resume indication from the MTP3 layer.
TUPMTP3PAUSE	MTP pause indication is delivered to the application when the TUP layer receives a resume indication from the MTP3 layer.

5 TUP service function reference

TUP service function summary

The TUP service consists of the following functions:

Function	Description
TUPADDRtoASCII	Changes a packed string of digits (appropriate for the information elements that require an address string) to an ASCII string of digits.
TUPASCIItoADDR	Changes an ASCII string of digits (to dial) to a packed structure of digits appropriate for the information elements that require an address string.
TUPConnectReq	Requests the establishment of a circuit switched connection.
TUPConnectResp	Signals the far exchange that an incoming call was answered.
	Sends connection status information to the far exchange during the connection establishment phase.
TUPGetApiStats	Retrieves congestion level activity statistics from the TUP service.
TUPReleaseReq	Clears or denies the establishment of a circuit switched connection.
TUPReleaseResp	Responds to a release indication from a far exchange.
TUPRetrieveMessage	

Using the TUP service function reference

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

Prototype	The prototype is followed by a listing of the function arguments. NMS data types include: • DWORD (8-bit unsigned) • S16 (16-bit signed) • U32 (32-bit unsigned) • U8 (8-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 TUP_SUCCESS or an error code. For asynchronous functions, a return value of TUP_SUCCESS (zero) indicates the function was initiated; subsequent events indicate the status of the operation.

TUPADDRtoASCII

Changes a packed string of digits (appropriate for the information elements that require an address string) to an ASCII string of digits.

Prototype

U8 NMSAPI **TUPADDRtoASCII** (TupTknStr ***addrSigs***, char ***ascii***)

Argument	Description
addrSigs	Pointer to the packed list.
ascii	Pointer to resulting ASCII string.

Return values

None.

TUPASCIItoADDR

Changes an ASCII string of digits (to dial) to a packed structure of digits appropriate for the information elements that require an address string.

Prototype

U8 NMSAPI **TUPASCIItoADDR** (char ***ascii***, TupTknStr ***addrSigs***, short ***length***)

Argument	Description
ascii	Pointer to ASCII string of digits.
addrSigs	Pointer to resulting packed list.
length	Length of the ASCII parameter.

Return values

None.

Details

TUPASCIItoADDR can be used with the LineID information element fields (calling address or original called address) and the AddrSigs information element.

TUPConnectReq

Requests the establishment of a circuit switched connection.

Prototype

DWORD NMSAPI **TUPConnectReq** (CTAHD ***ctahd***, U8 ***board***, TupSpId ***spId***, TupInstId ***suInstId***, TupInstId ***spInstId***, Bool ***cirSelFlg***, CirIdx ***circuit***, TupConEvnt ****conEvnt***)

Argument	Description
ctahd	Natural Access handle.
board	TX board number.
spId	TUP service access point.
suInstId	Service user instance ID.
spInstId	Service provider instance ID.
cirSelFlg	Circuit selection flag.
circuit	Circuit index used for this connection if ***cirSelFlg*** is set to true.
conEvnt	Pointer to the caller's connect event structure containing all parameters (IEs) relevant to establishing this connection.

Return values

Return value	Description
TUP_SUCCESS	
TUP_INVBOARD	***board*** is out of range.
TUP_NOTBOUND	Application failed to open the TUP service.
TUP_OSERROR	Error occurred accessing the driver.

Details

When successful, **TUPConnectReq** results in an initial address message (IAM, IAI) being sent to the far exchange. The TUP service generates an IAI message if any one of the cgPtyNum, origCdNum, and clUsrGrp IEs are filled. Otherwise, an IAM is generated.

If the TUP layer cannot successfully initiate the outgoing connection request (for example, due to network congestion or the requested circuit not being idle), it returns an asynchronous STATUS INDICATION event to the application with the cause value coded with the reason for the failure.

For more information, refer to *Establishing connections* on page 28 and *Handling failed connection requests* on page 32.

TUPConnectResp

Signals the far exchange that an incoming call was answered.

Prototype

DWORD NMSAPI **TUPConnectResp** (CTAHD ***ctahd***, U8 ***board***, TupSpId ***spId***, TupInstId ***suInstId***, TupInstId ***spInstId***, CirIdx ***circuit***, TupConRspEvnt ****conRspEvnt***)

Argument	Description
ctahd	Natural Access handle.
board	TX board number.
spId	TUP service access point.
suInstId	Service user instance ID.
spInstId	Service provider instance ID.
circuit	Circuit index with which this message is associated.
conRspEvnt	Pointer to the caller's connect event structure containing all parameters (IEs) included in the answer message.

Return values

Return value	Description
TUP_SUCCESS	
TUP_INVBOARD	***board*** is out of range.
TUP_NOTBOUND	Application failed to open the TUP service.
TUP_OSERROR	Error occurred accessing the driver.

Details

TUPConnectResp generates an answer (ANC, ANN, ANU) message to the far exchange. The only field that needs to be filled for this call is answerType in the conRspEvnt with ANU_NO_CHARGE, ANU_CHARGE, or ANU_UNQUALIFIED (not supported in China).

TUPConnectStatusReq

Sends connection status information to the far exchange during the connection establishment phase.

Prototype

DWORD NMSAPI **TUPConnectStatusReq** (CTAHD ***ctahd***, U8 ***board***, TupSpId ***spId***, TupInstId ***suInstId***, TupInstId ***spInstId***, CirIdx ***circuit***, TupCnStEvnt ****conStEvnt***, U8 ***evntType***)

Argument	Description
ctahd	Natural Access handle.
board	TX board number.
spId	TUP service access point.
suInstId	Service user instance ID.
spInstId	Service provider instance ID.
circuit	Circuit index with which this message is associated.
conStEvnt	Pointer to the caller's connect status event structure containing all parameters (IEs) included in the message to the far exchange.
evntType	Identifies the type of message sent to the far exchange. Refer to the Details section for a list of valid message types.

Return values

Return value	Description
TUP_SUCCESS	
TUP_INVBOARD	***board*** is out of range.
TUP_NOTBOUND	Application failed to open the TUP service.
TUP_OSERROR	Error occurred accessing the driver.
TUP_UNKEVENT	Unrecognized event type requested.

Details

The connection status information can be address complete, progress, information request, and so on. For more information, refer to *Establishing connections* on page 28.

The following table lists the valid message types:

Message type	**Description**
TUPADDRCMPLT	Address complete (ACM)
TUPFRWDTRSFR	Forward transfer (FOT)
TUPINFORMATION	Information (GSM response to GRQ)
TUPINFORMATREQ	Information request (GRQ)
TUPSUBSADDR	Subsequent address message (SAM, SAO)
TUPMALICIOUS	Malicious call signal (MAL)
TUPREANSWER	Re-answer signal (RAN)
TUPCALLCLEAR	Calling party clear (CCL)
TUPAUTOCONG	Auto-congestion message (ACC)
TUPMETERPULSE	Metering pulse message (MPM) - China only
TUPOPERATOR	Operator signal (OPR) - China only

TUPGetApiStats

Retrieves congestion level activity statistics from the TUP service.

Prototype

DWORD NMSAPI **TUPGetApiStats** (CTAHD ***ctahd***, TUPAPISTATS ***pstats***, U8 ***bReset***)

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td><b><i>ctahd</i></b></td><td>Natural Access handle.</td></tr>
<tr><td><b><i>pstats</i></b></td><td>Pointer to the buffer address where statistics are returned to the caller:
<pre>
typedef struct
{
   U32 qCount;   /* number of API messages currently
                     * queued to TUP layer   */
   U32 qPeak;   /* max number of API messages ever
                     * queued to TUP layer   */
   U32 txPending;   /* current number of outstanding transmit
                     * rqsts to TUP layer   */
   U32 txPendPeak; /* max number of transmit rqsts ever
                     * outstanding to TUP layer   */
   U32 txSuccess;   /* number of successful transmit requests
                     * completed   */
   U32 txFailed;   /* number of failed transmit requests   */
   U32 txLastErr;   /* error code from last failed
                     * transmit request   */
   U32 rxSuccess;   /* number of events received from TUP
                     * layer   */
   U8 apiQCongLvl; /* current outbound queue congestion
                     * level [0..3]   */
   U8 tupCongLvl; /* current TUP layer congestion
                     * level [0..3]   */
   U8 tupCongSrc; /* reason for TUP layer congestion   */
   U8 spare1;   /* spare for alignment   */
} TUPAPISTATS;
</pre></td></tr>
<tr><td><b><i>bReset</i></b></td><td>If non-zero, statistics are reset after returning the statistics to the application.</td></tr>
</table>

Return values

Return value	Description
TUP_SUCCESS	
CTAERR_INVALID_CTAHD	Invalid handle provided.

TUPReleaseReq

Clears or denies the establishment of a circuit switched connection.

Prototype

DWORD NMSAPI **TUPReleaseReq** (CTAHD ***ctahd***, U8 ***board***, TupSpId ***spId***, TupInstId ***suInstId***, TupInstId ***spInstId***, CirIdx ***circuit***, TupRelEvnt ****relEvnt***)

Argument	Description
ctahd	Natural Access handle.
board	TX board number.
spId	TUP service access point.
suInstId	Service user instance ID.
spInstId	Service provider instance ID.
circuit	Circuit index with which this message is associated.
relEvnt	Pointer to the caller's release event structure containing all parameters (IEs) included in the RELEASE message to the far exchange.

Return values

Return value	Description
TUP_SUCCESS	
TUP_INVBOARD	***board*** is out of range.
TUP_NOTBOUND	Application failed to open the TUP service.
TUP_OSERROR	Error occurred accessing the driver.

Details

TUPReleaseReq generates a clear forward message to the far exchange. The ***relEvnt*** parameter does not contain any relevant information for the TUP service.

For more information, refer to *Clearing connections* on page 30.

TUPReleaseResp

Responds to a release indication from a far exchange.

Prototype

DWORD NMSAPI **TUPReleaseResp** (CTAHD ***ctahd***, U8 ***board***, TupSpId ***spId***, TupInstId ***suInstId***, TupInstId ***spInstId***, CirIdx ***circuit***, TupRelEvnt ****relEvnt***)

Argument	Description
ctahd	Natural Access handle.
board	TX board number.
spId	TUP service access point.
suInstId	Service user instance ID.
spInstId	Service provider instance ID.
circuit	Circuit index with which this message is associated.
relEvnt	Pointer to the caller's release event structure containing all parameters (IEs) included in the release complete message to the far exchange.

Return values

Return value	Description
TUP_SUCCESS	
TUP_INVBOARD	***board*** is out of range.
TUP_NOTBOUND	Application failed to open the TUP service.
TUP_OSERROR	Error occurred accessing the driver.

Details

TUPReleaseResp sends a release guard (RLG) message to the far exchange and makes the circuit available for a new connection in the TUP circuit database. The ***relEvnt*** parameter does not contain any relevant information for the TUP service.

For more information, refer to *Clearing connections* on page 30.

TUPRetrieveMessage

Retrieves the next message from the TUP layer or optionally waits for a message if none is currently available.

Prototype

DWORD NMSAPI **TUPRetrieveMessage** (CTAHD ***ctahd***, TupAllSdus ****event***, TupRcvInfoBlk ****infoBlk***)

Argument	Description
ctahd	Natural Access handle.
event	Pointer to the address of the caller's event buffer where the received event (if any) is returned to the caller.

infoBlk	Pointer to the address of the caller's receive information block where information regarding the received event (if any) is returned to the caller:
	`typedef struct rcvInfoBlk`
	`{`
	`    U8 indType;         /* indic/confirm type                    */`
	`    U8 evntType;        /* event type for release, status, and`
	`                        /* connection status indications         */`
	`    SuId suId;          /* service user (SAP) id - all           */`
	`    TupInstId suInstId; /* caller's reference number - all       */`
	`    TupInstId spInstId; /* TUP's reference number - all          */`
	`    CirIdx circuit;     /* circuit index - all                   */`
	`} TupRcvInfoBlk;`

Return values

Return value	Description
TUP_SUCCESS	
TUP_NOMSG	No event messages waiting.
TUP_NOTBOUND	Application failed to open the TUP service.
TUP_RESOURCES	Could not allocate message buffer.

Details

Periodically call this function to receive events (messages) from the TUP layer.

When a message is received, **TUPRetrieveMessage** copies the event to the caller's event buffer and performs any necessary byte order translation to convert to the host's native byte ordering. Information about the event is returned to the caller in the ***infoBlk*** parameter.

The caller's event buffer must be large enough to accommodate any of the events, as defined by the TupAllSdus structure (union of all event structures). The actual event structure returned (which is a member of the union) depends on the value of the infoBlk.indType field returned, as shown in the following table:

Indication type	**Event structure employed**
EVTTUPCONCFM	TupConRspEvnt
EVTTUPCONIND	TupConEvnt
EVTTUPCNSTIND	TupCnStEvnt
EVTTUPRELIND	TupRelEvnt
EVTTUPRELCFM	TupRelEvnt
EVTTUPSTAIND	TupStaEvnt
EVTTUPMGTSTAIND	TupStaEvnt

The indication type (indType) identifies the event received and is coded to one of the following values:

Indication type	**Hex**	**Description**
EVTTUPCONCFM	0x0D	Connect confirm
EVTTUPCONIND	0x0E	Connect indication
EVTTUPCNSTIND	0x5A	Connect status indication
EVTTUPRELIND	0x5E	Connect release indication
EVTTUPRELCFM	0x5D	Connect release confirmation
EVTTUPSTAIND	0x7A	Status indication
EVTTUPMGTSTAIND	0x7B	Local management status indication

The event type (evntType) identifies the actual message received for status and connection status indications. It is coded to one of the values in the following tables.

For connection status indications:

Event type	Description
TUPADDRCMPLT	Address complete (ACM)
TUPFRWDTRSFR	Forward transfer (FOT)
TUPINFORMATION	Information (response to INFORMATREQ, GSM)
TUPINFORMATREQ	Information request (GRQ)
TUPSUBSADDR	Subsequent address message (SAM)
TUPSUBSADDRONE	Single subsequent address (SAO)
TUPREANSWER	Re-answer signal (RAN)
TUPCALLCLEAR	Calling party clear (CCL)
TUPAUTOCONG	Auto-congestion message (ACC)
TUPOPERATOR	Operator signal (OPR) - China only
TUP METERPULSE	Metering pulse message (MPM) - China only
TUPMALICIOUS	Malicious call signal (MAL) - China only

For status indications (including local management indications):

Event type	Description
TUPREATTEMPT	Re-attempt indication
TUPERRORIND	Error indication
TUPCONFUSION	Confusion indication
TUPCONTCHK	Continuity check (CCR)
TUPCONTREP	Continuity report (COT)
TUPCONFAIL	Continuity report (CCF)
TUPCIRBLKREQ	Circuit block request (BLO)
TUPCIRUNBLKREQ	Circuit unblock request (UBL)
TUPCIRRESREQ	Circuit reset request (RSC)
TUPCRMGRPBLKREQ	Maintenance circuit group block request (MGB)
TUPCRMGRPUNBLKREQ	Maintenance circuit group unblock request (MGU)
TUPCRHGRPBLKREQ	Hardware circuit group block request (HGB)
TUPCRHGRPUNBLKREQ	Hardware circuit group unblock request (HGU)
TUPCRSGRPBLKREQ	Software circuit group block request (SGB)
TUPCRSGRPUNBLKREQ	Software circuit group unblock request (SGU)
TUPCIRGRPRESREQ	Circuit group reset request (GRS)

Event type	**Description**
TUPSWITCHCONG	Switch equipment congestion (CNG)
TUPCIRCUITCONG	Circuit group congestion (CGC)
TUPNATIONALCONG	Nation network congestion (NNC)
TUPADDRESSINCOMP	Address incomplete (ADI)
TUPCALLFAILURE	Call failure (CFL)
TUPSUBBUSY	Subscriber busy (SSB)
TUPUNALLOCATEDNUM	Unallocated number (UNN)
TUPOUTOFSERVICE	Line out of service (LOS)
TUPSENDSPECIAL	Send a special tone (SST)
TUPACCESSBARRED	Access barred (ACB)
TUPNODIGITAL	Digital path not provided (DPN)
TUPEXTENDEDINFO	Extended unsuccessful message (EUM)
TUPCLEARBKW	Clear backwards (CBK)
TUPMISDIALEDPREFIX	Misdialed trunk prefix (MPR) - not supported in China
TUPSUBLOCALBUSY	Subscriber local busy (SLB) - China only
TUPSUBTOLLBUSY	Subscriber toll busy (STB) - China only
TUPDUALSEIZURE	Call failed due to incoming call

The application must save the service provider instance ID (spInstId) field from the first event received from TUP for each connection and use it in subsequent requests associated with that connection.

For more information, refer to *Operating system specific considerations* on page 21.

TUPStatusReq

Sends a global or circuit-specific message to the far exchange.

Prototype

DWORD NMSAPI **TUPStatusReq** (CTAHD ***ctahd***, U8 ***board***, TupSpId ***spId***, TupInstId ***suInstId***, TupInstId ***spInstId***, CirIdx ***circuit***, U8 ***evntType***, TupStaEvnt ****staEvnt***)

Argument	Description
ctahd	Natural Access handle.
board	TX board number.
spId	TUP service access point.
suInstId	Service user instance ID.
spInstId	Service provider instance ID.
circuit	For circuit-specific requests, the circuit index with which this request is associated. For circuit group specific requests, this argument must identify one member of the circuit group.
evntType	Type of status request. Refer to the Details section for a list of valid status request types.
staEvnt	Pointer to the caller's status event structure containing all parameters (IEs) included in the message to the far exchange.

Return values

Return value	Description
TUP_SUCCESS	
TUP_INVBOARD	***board*** is out of range.
TUP_NOTBOUND	Application failed to open the TUP service.
TUP_OSERROR	Error occurred accessing the driver.
TUP_UNKEVENT	Unrecognized event type requested.

Details

The following table lists the valid status request types (***evntType***):

Status request type	Description
TUPCONTCHK	Continuity check (CCR)
TUPCONTREP	Continuity report (COT)
TUPCONFAIL	Continuity report (CCF)
TUPCIRBLKREQ	Circuit block request (BLO)
TUPCIRUNBLKREQ	Circuit unblock request (UBL)
TUPCIRRESREQ	Circuit reset request (RSC)

Status request type	Description
TUPCRMGRPBLKREQ	Maintenance circuit group block request (MGB)*
TUPCRMGRPUNBLKREQ	Maintenance circuit group unblock request (MGU)*
TUPCRHGRPBLKREQ	Hardware circuit group block request (HGB)*
TUPCRHGRPUNBLKREQ	Hardware circuit group unblock request (HGU)*
TUPCRSGRPBLKREQ	Software circuit group block request (SGB)*
TUPCRSGRPUNBLKREQ	Software circuit group unblock request (SGU)*
TUPCIRGRPRESREQ	Circuit group reset request (GRS)
TUPSWITCHCONG	Switch equipment congestion (SEC)
TUPCIRCUITCONG	Circuit group congestion (CGC)
TUPNATIONALCONG	Nation network congestion (NNC)
TUPADDRESSINCOMP	Address incomplete (ADI)
TUPCALLFAILURE	Call failure (CFL)
TUPSUBBUSY	Subscriber busy (SSB)
TUPUNALLOCATEDNUM	Unallocated number (UNN)
TUPOUTOFSERVICE	Line out of service (LOS)
TUPSENDSPECIAL	Send special tone (SST)
TUPACCESSBARRED	Access barred (ACB)
TUPNODIGITAL	Digital path not provided (DPN)
TUPEXTENDEDINFO	Extended unsuccessful message (EUM)
TUPCLEARBKW	Clear backwards (CBK)
TUPMISDIALEDPREFIX	Misdialed trunk prefix (MPR) - not supported in China
TUPSUBLOCALBUSY	Subscriber local busy (SLB) - China only
TUPSUBTOLLBUSY	Subscriber toll busy (STB) - China only

When sending one of the status request events marked with an asterisk (*), populate the range information element in ***staEvnt***. If all circuits in the predefined group should be acted upon, set the range field to zero (0) and ignore the ***statusMap***. If some of the circuits should not be acted upon or this is not a predefined group, set range to the number of circuits that can be acted upon. Then set the ***statusMap*** array as follows: set an element indexed into this array to one (1) if this circuit should be acted upon, or set it to zero (0) if it should not be acted upon. The first element in the array maps to the circuit parameter sent to **TUPStatusReq**. The remainder of the array maps to the consecutive circuits in the defined group. Refer to the TUP specifications for more information.

6 Managing TUP tasks on the board

TUP management overview

TUP management functions manage the following TUP tasks on the board:

- Configuration
- Managing circuits
- Retrieving statistics and status

Any application that uses TUP management must start with a call to **TUPInitMgmtAPI**. **TUPTermMgmtAPI** properly closes the application.

All calls are blocking while waiting for a response from the board's TUP task. An error code is returned when a call is unsuccessful.

Configuration

TUP management consists of the following types of configuration functions:

Function type	Description
Init	Fills the configuration structure.
Set	Sends a configuration to the board.
Get	Retrieves the configuration from the board.

Refer to the TUP management function reference section for information about the function calls and the default and possible values for configuration elements.

General configuration

The general configuration must be done before any other configurations. Parameters set up in the general configuration include the number of service access points, the maximum number of circuits and groups allowed, and alarm level. A call to **TUPInitGenCfg** fills in all of the appropriate elements with default values for the application in a TUPGenCfg structure. A call to **TUPSetGenCfg** then sends the configuration to the board. **TUPGetGenCfg** retrieves this configuration from the board.

USAP and NSAP configuration

After the general configuration, configure the service access points above and below the TUP task on the board. Calls to **TUPInitUSapCfg** and **TUPInitNSapCfg** fill in the fields in the appropriate structures with default values for the application, which then can call **TUPSetUSapCfg** and **TUPSetNSapCfg** to send the configurations to the board. These configurations must not be changed after initialization of the task. The fields in the user service access point section define switch type (either ITU-T or the China variant) and the user queue congestion onset and abatement levels. The network service access point configuration also defines the MTP network service access point with which to bind this NSAP, as well as the point code. **TUPGetUSapCfg** and **TUPGetNSapCfg** retrieve these configurations from the board.

Circuits and groups configuration

After the general configuration and the service access point configurations, configure the circuits for the TUP task to manage. First, calls to **TUPInitCircCfg** fill in some of the fields with default values for the application. After filling in and adjusting the remaining fields, the application can call **TUPSetCircCfg** to send the configurations to the board. These configurations can be changed after initialization of the task. The most important fields in circuit and group configuration are:

- Destination point code for this circuit set
- Circuit index and the circuit identification code of the first circuit in the set
- Number of circuits in the set
- Switch type (ITU-T or China) and the circuit group ID

The remaining circuit identification codes (CIC) are managed and used through the circuit index that correlates to the CIC. For example, if the CIC field is 10 and the first circuit index is 20 in a five circuit set, manage or make a call on CIC 13 by using circuit index 23 in the function calls. If the group ID field in the configuration request is non-zero, the circuit set is treated as a predefined group. **TUPGetCircCfg** retrieves this configuration from the board.

Use alternate originating point code for this circuit set when configuring the board to act as multiple originating point codes (OPCs). The OPC must be properly configured in MTP for the new TUP OPC to work. By default, configuration uses the TUP general configuration originating point code. For more information on configuring multiple OPCs, see the *NMS SS7 Configuration Manual*.

Note: TUP supports pre-defined groups that allow applications to send actions to well-known sets of circuits. Confirm that the carrier in which the application will be installed supports pre-defined groups. Otherwise, the application needs to use dynamic groups only.

Managing circuits

The TUP management functions allow circuits and circuit groups to be:

- Reset (**TUPResetCircuit**, **TUPResetGroup**)
- Blocked (**TUPBlockCircuit**, **TUPBlockGroup**)
- Unblocked (**TUPUnblockCircuit**, **TUPUnblockGroup**)
- Deleted (**TUPDeleteCircuit**, **TUPDeleteGroup**)

These actions generate a management status indication (EVTTUPMGTSTAIND) event to the application instead of a regular status indication event (EVTTUPSTAIND).

Retrieving statistics and status

The TUP task keeps statistics and status for a management application to query. The statistics are available by circuit (**TUPGetCircStats**), circuit group (**TUPGetGrpStats**), and network service access point (**TUPGetNSapStats**). Status is available by circuit (**TUPGetCircStatus**) and circuit group (**TUPGetGrpStatus**).

7 TUP management function reference

TUP management function summary

NMS TUP consists of the following management functions:

- Configuration functions
- Control functions
- Statistics functions
- Status functions

Configuration functions

Function	Description
TUPGetCircCfg	Gets circuit configuration from the TX board.
TUPGetGenCfg	Retrieves the basic configuration from the TX board.
TUPGetGrpCfg	Retrieves the current, pre-defined circuit group configuration.
TUPGetNSapCfg	Retrieves the basic network service access point configuration from the TX board.
TUPGetUSapCfg	Retrieves the basic user service access point configuration from the TX board.
TUPInitCircCfg	Initializes a buffer that can be passed to **TUPSetCircCfg**.
TUPInitGenCfg	Builds a basic configuration buffer that can be passed to **TUPSetGenCfg**.
TUPInitGrpCfg	Initializes a group configuration structure.
TUPInitNSapCfg	Builds a basic configuration buffer that can be passed to **TUPSetNSapCfg**.
TUPInitUSapCfg	Builds a basic configuration buffer that can be passed to **TUPSetUSapCfg**.
TUPSetCircCfg	Sets circuit configurations to the TX board.
TUPSetGenCfg	Sends a basic configuration buffer to the TX board.
TUPSetNSapCfg	Sends a basic network service access point configuration buffer to the TX board.
TUPSetUSapCfg	Sends a basic user service access point configuration buffer to the TX board.

Control functions

Function	Description
TUPAlarmControl	Sends a request to control alarms.
TUPBlockCircuit	Sends a request to block the given circuit.
TUPBlockGroup	Sends a request to block the given group of circuits.
TUPDeleteCircuit	Sends a request to delete the given circuit.
TUPDeleteGroup	Sends a request to delete the given group of circuits.
TUPInitMgmtAPI	Initializes TUP management and opens a channel to the TX board.
TUPQuietReset	Sends a request to the board to reset all circuits without generating TUP traffic.
TUPResetCircuit	Sends a request to reset the given, pre-defined circuit.
TUPResetGroup	Sends a request to reset the given group of circuits.
TUPTermMgmtAPI	Closes TUP management and the channel to the TX board.
TUPTraceControl	Sends a request to enable or disable tracing of TUP protocol messages.
TUPUnblockCircuit	Sends a request to unblock the given circuit.
TUPUnblockGroup	Sends a request to unblock the given group of circuits.

Statistics functions

Function	Description
TUPGetCircStats	Sends a request to retrieve the statistics for a given circuit.
TUPGetGrpStats	Sends a request to retrieve the statistics for a given, pre-defined circuit group.
TUPGetNSapStats	Sends a request to retrieve the statistics for a given network service access point.

Status functions

Function	Description
TUPGetCircStatus	Sends a request to retrieve the status for a given circuit.
TUPGetGrpStatus	Sends a request to retrieve the status for a given, pre-defined circuit group.

Using the TUP management function reference

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

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

TUPAlarmControl

Sends a request to control alarms.

Prototype

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

Argument	Description
board	TX board number.
flags	Bitmap of alarm control. Reserved for future use.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

TUPBlockCircuit

Sends a request to block the given circuit.

Prototype

S16 **TUPBlockCircuit** (U8 ***board***, CirIdx ***circID***)

Argument	Description
board	TX board number.
circID	Circuit index to block.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

TUPBlockGroup

Sends a request to block the given group of circuits.

Prototype

S16 **TUPBlockGroup** (U8 ***board***, U16 ***groupID***, U8 ***blockTyp***)

Argument	Description
board	TX board number.
groupID	Group ID to block.
blockTyp	Type of block. Possible values are TUP_MBLOCK, TUP_HBLOCK, or TUP_SBLOCK.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

TUPDeleteCircuit

Sends a request to delete the given circuit.

Prototype

S16 **TUPDeleteCircuit** (U8 ***board***, CirIdx ***circID***, U32 ***count***)

Argument	Description
board	TX board number.
circID	Circuit index at which to start deleting.
count	Number of circuits to delete.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

Details

Set ***circId*** and ***count*** to 0 to delete all circuits that are not part of a predefined group. Call **TUPDeleteGroup** with groupId 0 before calling **TUPDeleteCircuit** to delete all circuits.

TUPDeleteGroup

Sends a request to delete the given group of circuits.

Prototype

S16 **TUPDeleteGroup** (U8 ***board***, U16 ***groupID***)

Argument	Description
board	TX board number.
groupID	Group ID to delete. Set to 0 to delete all groups.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

TUPGetCircCfg

Gets circuit configuration from the TX board.

Prototype

S16 **TUPGetCircCfg** (U8 ***board***, TUPCircCfg ****cfg***, CirIdx ***circID***)

Argument	Description
board	TX board number.
cfg	Pointer to the circuit configuration structure to fill. Refer to **TUPInitUSapCfg**.
circId	Circuit index for which to retrieve configuration.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

TUPGetCircStats

Sends a request to retrieve the statistics for a given circuit.

Prototype

S16 **TUPGetCircStats** (U8 ***board***, CirIdx ***circID***, TUPCircStats ****stats***, U8 ***reset***)

Argument	Description
board	TX board number.
circID	Circuit index for which to retrieve statistics.
stats	Pointer to structure to place retrieved statistics. Refer to the Details section for more information.
reset	Set this to non-zero to reset the current statistics.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

Details

The TupCircStats structure appears as follows:

```
typedef struct _TupCircStats
{
    DateTime dt;      /* date and time (Not Used)             */
    Duration dura;    /* duration (Not Used)                  */
    S32 blockTx;      /* Circuit Blocking Transmitted         */
    S32 blockAckTx;   /* Circuit Blocking Ack Transmitted     */
    S32 unblockTx;    /* Circuit Unblocking Transmitted       */
    S32 unblockAckTx; /* Circuit Unblocking Ack Transmitted   */
    S32 cirResTx;     /* Circuit Reset Transmitted            */
    S32 adrCmpltTx;   /* Address complete transmitted         */
    S32 answerTx;     /* Answer transmitted                   */
    S32 contiTx;      /* Continuity transmitted               */
    S32 conChkReqTx;  /* Continuity Check Request transmitted */
    S32 fotTx;        /* Forward transmitted                  */
    S32 clfTx;        /* Release transmitted                  */
    S32 cclTx;        /* Caller clear transmitted             */
    S32 rlgTx;        /* Release Complete transmitted         */
    S32 cbkTx;        /* clear back transmitted               */
    S32 ranTx;        /* Reanswer transmitted                 */
    S32 oprTx;        /* operator transmitted                 */
    S32 mpmTx;        /* meter pulse transmitted              */
    S32 accTx;        /* auto cong level transmitted          */
    S32 ubmTx;        /* UBM txed, including SLB and STB      */
    S32 initAdrTx;    /* Initial Address transmitted          */
    S32 infoTx;       /* Info transmitted                     */
    S32 infoReqTx;    /* Info Request transmitted             */
    S32 subsAdrTx;    /* Subsequent Address transmitted       */
    S32 malTx;        /* Malicious Call Print transmitted     */
    S32 chargeTx;     /* Charge transmitted                   */
    S32 blockRx;      /* Circuit Blocking Received            */
    S32 blockAckRx;   /* Circuit Blocking Ack Received        */
    S32 unblockRx;    /* Circuit Unblocking Received          */
    S32 unblockAckRx; /* Circuit Unblocking Ack Received      */
    S32 cirResRx;     /* Circuit Reset Received               */
    S32 adrCmpltRx;   /* Address complete received            */
    S32 answerRx;     /* Answer received                      */
    S32 contiRx;      /* Continuity received                  */
    S32 conChkReqRx;  /* Continuity Check Request received    */
    S32 fotRx;        /* Forward received                     */
    S32 clfRx;        /* Release received                     */
    S32 cclRx;        /* Caller clear received                */
    S32 rlgRx;        /* Release Complete received            */
    S32 cbkRx;        /* clear back received                  */
    S32 ranRx;        /* Reanswer received                    */
    S32 oprRx;        /* operator received                    */
    S32 mpmRx;        /* meter pulse received                 */
    S32 accRx;        /* auto cong level received             */
    S32 ubmRx;        /* UBM received, including SLB/STB      */
    S32 initAdrRx;    /* Initial Address received             */
    S32 infoReqRx;    /* Info Request received                */
    S32 infoRx;       /* Info received                        */
    S32 subsAdrRx;    /* Subsequent Address received          */
    S32 malRx;        /* Malicious Call Print received        */
    S32 chargeRx;     /* Charge received                      */
} TupCircStats;
```

TUPGetCircStatus

Sends a request to retrieve the status for a given circuit.

Prototype

S16 **TUPGetCircStatus** (U8 ***board***, CirIdx ***circID***, TUPCircStatus ****status***)

Argument	Description
board	TX board number.
circID	Circuit index for which to retrieve status.

status	Pointer to structure where retrieved status is placed.
	`typedef struct _TupCircStatus /* TUP Circuit status            */`
	`{`
	`    DateTime dt;               /* date and time                 */`
	`    U16 circuitState;          /* circuit machine state         */`
	`    U16 callState;             /* circuit call processing state */`
	`} TupCircStatus;`
	Refer to the Details section for more information.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

Details

The circuitState element is coded as follows:

0x200	Initial
0x201	Idle
0x202	Idle, locally blocked
0x203	Idle, remotely blocked
0x204	Idle, remotely blocked, locally blocked
0x205	Busy incoming
0x206	Busy incoming, locally blocked
0x207	Busy incoming, remotely blocked
0x208	Busy incoming, remotely blocked, locally blocked
0x209	Busy outgoing
0x20A	Busy outgoing, locally blocked
0x20B	Busy outgoing, remotely blocked
0x20C	Busy outgoing, remotely blocked, locally blocked
0x20D	Out of service locally
0x20E	Out of service locally, locally blocked
0x20F	Out of service locally, remotely blocked
0x210	Out of service locally, remotely blocked, locally blocked
0x211	Out of service remotely
0x212	Out of service remotely, locally blocked
0x213	Out of service remotely, remotely blocked
0x214	Out of service remotely, remotely blocked, locally blocked
0x215	Out of service BOTH
0x216	Out of service BOTH, locally blocked
0x217	Out of service BOTH, remotely blocked
0x218	Out of service BOTH, remotely blocked, locally blocked

The callState element is coded as follows:

Inbound busy circuit:	
0x300	Initial state
0x301	Wait CLF state, in call
0x302	Wait info state
0x303	Wait answer state
0x304	Wait ACM state
0x305	Wait local RLG state
0x306	Wait COT state
0x307	Wait CCR state
0x308	Wait CLF to end cont check state
0x309	Dead state
Outgoing busy circuit:	
0x400	Initial state
0x401	Wait RLG state
0x402	Wait release state
0x403	Wait info state
0x404	Wait answer state
0x405	Wait ACM state
0x406	Wait local COT state
0x407	Wait local CONT CLF state
0x408	Wait next CCR state
0x409	Dead state

TUPGetGenCfg

Retrieves the basic configuration from the TX board. Refer to *Configuration* on page 60 for more information.

Prototype

S16 **TUPGetGenCfg** (U8 ***board***, TUPGenParms ***cfg***)

Argument	Description
board	TX board number from which to get the configuration.
cfg	Pointer to configuration structure to fill. Refer to **TUPInitGenCfg**.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

TUPGetGrpCfg

Retrieves the current, pre-defined circuit group configuration.

Prototype

S16 **TUPGetGrpCfg** (U8 ***board***, TupGrpCfg ****cfg***, U16 ***groupID***)

Argument	Description
board	TX board number.
cfg	Pointer to the configuration structure to fill. Refer to **TUPInitGrpCfg**.
groupID	Group ID for which to get the configuration.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

TUPGetGrpStats

Sends a request to retrieve the statistics for a given, pre-defined circuit group.

Prototype

S16 **TUPGetGrpStats** (U8 ***board***, U16 ***groupID***, TUPGrpStats ****stats***, U8 ***reset***)

Argument	Description
board	TX board number.
groupID	Group number from which to retrieve statistics.
stats	Pointer to structure in which to place retrieved statistics. Refer to the Details section for more information.
reset	Set this to non-zero to reset the current statistics.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

Details

The TUPGrpStats structure appears as follows:

```
typedef struct _TupGrpStats
{
    DateTime dt;          /* date and time (Not Used)            */
    Duration dura;        /* duration (Not Used)                 */
    S32 cirHwGrBlockRx;   /* HW Circuit Group Blocking Received  */
    S32 cirHwGrBlockAckRx;
                          /* HW Circuit Group Blocking Ack Rx    */
    S32 cirHwGrUnBlockRx;
                          /* HW Circuit Grp Unblocking Received  */
    S32 cirHwGrUnBlockAckRx;
                          /* HW Circuit Group Unblocking Ack Rx  */
    S32 cirSwGrBlockRx;   /* SW Circuit Group Blocking Received  */
    S32 cirSwGrBlockAckRx;
                          /* SW Circuit Group Blocking Ack Rx    */
    S32 cirSwGrUnBlockRx;
                          /* SW Circuit Grp Unblocking Received  */
    S32 cirSwGrUnBlockAckRx;
                          /* SW Circuit Group Unblocking Ack Rx  */
    S32 cirMGrBlockRx;    /* Maint Circuit Group Blocking Rx     */
    S32 cirMGrBlockAckRx; /* Maint Circuit Grp Blocking Ack Rx   */
    S32 cirMGrUnBlockRx;  /* Maint Circuit Grp Unblocking Rx     */
    S32 cirMGrUnBlockAckRx;
                          /* Maint Circuit Grp Unblocking Ack Rx */
    S32 cirGrResRx;       /* Circuit Group Reset Received        */
    S32 cirGrResAckRx;    /* Circuit Group Reset Ack Received    */
    S32 cirHwGrBlockTx;   /* HW Circuit Group Blocking Tx        */
    S32 cirHwGrBlockAckTx;
                          /* HW Circuit Group Blocking Ack Tx    */
    S32 cirHwGrUnBlockTx; /* HW Circuit Grp Unblocking Tx        */
    S32 cirHwGrUnBlockAckTx;
                          /* HW Circuit Group Unblocking Ack Tx  */
    S32 cirSwGrBlockTx;   /* SW Circuit Group Blocking Tx        */
    S32 cirSwGrBlockAckTx;
                          /* SW Circuit Group Blocking Ack Tx    */
    S32 cirSwGrUnBlockTx; /* SW Circuit Grp Unblocking Tx        */
    S32 cirSwGrUnBlockAckTx;
                          /* SW Circuit Group Unblocking Ack Tx  */
    S32 cirMGrBlockTx;    /* Maint Circuit Group Blocking Tx     */
    S32 cirMGrBlockAckTx; /* Maint Circuit Grp Blocking Ack Tx   */
    S32 cirMGrUnBlockTx;  /* Maint Circuit Grp Unblocking Tx     */
    S32 cirMGrUnBlockAckTx;
                          /* Maint Circuit Grp Unblocking Ack Tx */
    S32 cirGrResTx;       /* Circuit Group Reset Tx              */
    S32 cirGrResAckTx;    /* Circuit Group Reset Ack Tx          */
} TupGrpStats;
```

TUPGetGrpStatus

Sends a request to retrieve the status for a given, pre-defined circuit group.

Prototype

S16 **TUPGetGrpStatus** (U8 ***board***, U16 ***grpID***, TUPGrpStatus ***status***)

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td><b><i>board</i></b></td><td>TX board number.</td></tr>
<tr><td><b><i>grpID</i></b></td><td>Group from which to retrieve status.</td></tr>
<tr><td><b><i>status</i></b></td><td>Pointer to the following structure in which to place the retrieved status:
<pre>
typedef struct _TupGrpStatus
                     /* TUP Group status   */
{
    DateTime dt;   /* date and time (Not Used)   */
    U32 altOpc   /* originating point code   */
    U32 dpc;   /* destination point code   */
    U16 cic;   /* starting CIC   */
    U8 range;   /* range value, 0 if predefined   */
    U8 fill;   /* unused   */
    U16 count;   /* number of circuits in group   */
    U8 ocHMask[32]; /* status mask of local h/w block   */
    U8 ocMMask[32]; /* status mask of local maintenance block */
    U8 ocSMask[32]; /* status mask of local s/w block   */
    U8 remHMask[32]; /* status mask of local h/w block   */
    U8 remMMask[32]; /* status mask of local maintenance block */
    U8 remSMask[32]; /* status mask of local s/w block   */
} TupGrpStatus;
</pre></td></tr>
</table>

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

TUPGetNSapCfg

Retrieves the basic network service access point configuration from the TX board. Refer to *USAP and NSAP configuration* on page 60 for more information.

Prototype

S16 **TUPGetNSapCfg** (U8 ***board***, TUPNSapCfg ****cfg***, U16 ***sapID***)

Argument	Description
board	TX board number.
cfg	Pointer to the network service access point structure to fill. Refer to **TUPInitNSapCfg**.
sapID	Service access point ID for which to retrieve configuration.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

TUPGetNSapStats

Sends a request to retrieve the statistics for a given network service access point.

Prototype

S16 **TUPGetNSapStats** (U8 ***board***, S16 ***sapID***, TUPNSapStats ****stats***, U8 ***reset***)

Argument	Description
board	TX board number.
sapID	Network SAP for which to retrieve statistics.

stats	Pointer to the structure to place the retrieved statistics:
	`typedef struct _TupNSapStats`
	`{`
	`    DateTime dt;     /* date and time (Not Used)       */`
	`    Duration dura;   /* duration (Not Used)            */`
	`    S32 pktTx;       /* Number of transmitted packets */`
	`    S32 pktRx;       /* Number of received packets    */`
	`} TupNSapStats;`

reset	Set this to non-zero to reset the current statistics.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

TUPGetUSapCfg

Retrieves the basic user service access point configuration from the TX board. Refer to *USAP and NSAP configuration* on page 60 for more information.

Prototype

S16 **TUPGetUSapCfg** (U8 ***board***, TUPUSapCfg ****cfg***, U16 ***sapID***)

Argument	Description
board	TX board number.
cfg	Pointer to the user service access point structure to fill. Refer to **TUPInitUSapCfg**.
sapID	User service access point ID for which to retrieve configuration.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

TUPInitCircCfg

Initializes a buffer that can be passed to **TUPSetCircCfg**.

Prototype

void **TUPInitCircCfg** (TUPCircCfg ***cfg**, CirIdx ***circId***, U32 ***opc***, U32 ***dpc***)

Argument	Description
cfg	Pointer to the configuration structure to fill. Refer to the Details section for more information.
circId	Circuit index from which to start.
opc	Originating point code of this node for this circuit.
dpc	Destination point code of the node at the destination end of these circuits.

Return values

None.

Details

The elements of the TUPCircCfg structure are initialized with the following:

```
typedef struct _TUPCircCfg
{
    circuitId    circId argument
                        /* First circuit index in group           */
    cic          0      /* First Circuit ID code in group         */
    altOrgPointCode opc argument
                        /* Originating Point Code                 */
    dstPointCode    dpc argument
                        /* Destination point code                 */
    numCircs     1      /* Number of circuits in group            */
    groupId      0      /* Non-zero defines a pre-defined
                        /* group for group actions                */
    nsapId       0      /* NSAP from TUP config to use            */
    switchType   TUP_SW_ITU
                        /* Switch type for these circuits         */
    t1.value     15     /* Time to wait for COT(10-15)            */
    t1.enable    TRUE
    t2.value     30     /* Time to wait for ACM (20-30)           */
    t2.enable    FALSE
    t3.value     15     /* Time to wait for CLF after UBM  (4-15) */
    t3.enable    TRUE
    t4.value     15     /* Time to wait for CLF after CFL (4-15)  */
    t4.enable    TRUE
    t5.value     60     /* Time to stop sending CFL (60)          */
    t5.enable    TRUE
    t6.value     15       /* Time to wait for RLG (4-15)          */
    t6.enable    TRUE
    t7.value     60       /* Time to stop sending CLF (60)        */
    t7.enable    TRUE
    t8.value     2        /* Time to wait for backward check tone
                          /* (£ 2)                                 */
    t8.enable    TRUE
    t9.value     5        /* Time to delay 1st time continuity
                          /* check (1-10)                         */
    t9.enable    TRUE
    t10.value    180      /* Time to delay CCR (60-180)           */
    t10.enable   TRUE
    t11.value    60       /* Time to wait to alert after block
                          /* signal (60)                          */
    t11.enable   FALSE
    t12.value    15       /* Time to wait for response to block
                          /* signal (4-15)                        */
    t12.enable   TRUE
    t13.value    60       /* Time to wait to alert no response to
                          /* initial blocking signal (60)         */
    t13.enable   TRUE
    t14.value    60       /* Time to wait to repeat BLO(60)       */
    t14.enable   TRUE
    t15.value    15       /* Time to wait for response to
                          /* unblock (4-15)                       */
    t15.enable   TRUE
    t16.value    60       /* Time to wait to alert no response to
                          /* initial unblocking signal (60)       */
    t16.enable   TRUE
    t17.value    60       /* Time to wait to repeat UBL(60)       */
    t17.enable   TRUE
    t18.value    15       /* Time to wait for reset circuit
                          /* response (4-15)                      */
    t18.enable   TRUE
    t19.value    60       /* Time to wait to resend RSC(60)       */
    t19.enable   TRUE
} TUPCircCfg;
```

For more information, refer to *Circuits and groups configuration* on page 60.

TUPInitGenCfg

Builds a basic configuration buffer that can be passed to **TUPSetGenCfg**.

Prototype

void **TUPInitGenCfg** (TUPGenParms ***cfg**)

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td><b><i>cfg</i></b></td><td>Pointer to the configuration structure to fill.
The TUPGenParms structure contains these elements and is filled as follows:
<pre>
typedef struct _TUPGenParms
{
    maxSaps   1   /* Maximum number of user SAPs   */
    maxNetSaps   1   /* Maximum # interfaces to MTP3   */
    maxCircuits 96 /* Maximum # of circuits to manage   */
    maxCircGrp   32 /* Maximum # circuit groups to manage   */
    alarmLevel   1   /* Alarm level to filter at (1-4)   */
    maxOpcs   16 /* Maximum # of OPC's   */
    maxDpcs   16 /* Maximum number of DPC's   */
    dfltGrpCfg   /* see TUPInitGrpCfg   */
} TUPGenParms;
</pre></td></tr>
</table>

Return values

None.

Details

All unlisted elements in the TUPGenParms structure are set for the proper operation of the TUP stack and must not be changed.

Refer to **TUPInitGrpCfg** for the dfltGrpCfg element settings. Refer to *Configuration* on page 60 for more information about general configuration.

TUPInitGrpCfg

Initializes a group configuration structure.

Prototype

void **TUPInitGrpCfg** (TupGrpCfg ***cfg**, U16 ***count***)

Argument	Description
cfg	Pointer to the configuration structure to fill.
count	Number of circuits.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

Details

The dfltGrpCfg element settings are:

```
typedef struct _TUPGrpCfg
{
    count             /* The count parameter
    range       0     /* Predefined groups can use 0            */
    t20.enable  TRUE  /* Time to wait to Tx second GRS (5)      */
    t20.value   5
    t21.enable  TRUE  /* Time to wait for GRA (4-15)            */
    t21.value   15
    t22.enable  TRUE  /* Time to wait to Tx another GRS (60)    */
    t22.value   60
    t23.enable  TRUE  /* Time to wait to send 2nd MGB (5)       */
    t23.value   5
    t24.enable  TRUE  /* Time to wait to Tx 2nd MGU (5)         */
    t24.value   5
    t25.enable  TRUE  /* Time to wait to alert after MGB (300)  */
    t25.value   300
    t26.enable  TRUE  /* Time to wait for MBA (4-15)            */
    t26.value   15
    t27.enable  TRUE  /* Time to wait to resend MGB (60)        */
    t27.value   60
    t28.enable  TRUE  /* Time to wait MUA (4-15)                */
    t28.value   15
    t29.enable  TRUE  /* Time to send another MGU (60)          */
    t29.value   60
    t30.enable  TRUE  /* Time to wait to send 2nd HGB (5)       */
    t30.value   5
    t31.enable  TRUE  /* Time to wait to send 2nd HGU (5)       */
    t31.value   5
    t32.enable  TRUE  /* Time to wait for HBA (4-15)            */
    t32.value   15
    t33.enable  TRUE  /* Time to wait to send another HGB (60)  */
    t33.value   60
    t34.enable  TRUE  /* Time to wait for HUA (4-15)            */
    t34.value   15
    t35.enable  TRUE  /* Time to wait to send another HGU (60)  */
    t35.value   60
    t36.enable  TRUE  /* Time to wait to send 2nd SGB (5)       */
    t36.value   5
    t37.enable  TRUE  /* Time to wait to send 2nd SGU (5)       */
    t37.value   5
    t38.enable  TRUE  /* Time to wait for SBA (4-15)            */
    t38.value   15
    t39.enable  TRUE  /* Time to wait to send another SGB (60)  */
    t39.value   60
    t40.enable  TRUE  /* Time to wait for SUA (4-15)            */
    t40.value   15
    t41.enable  TRUE  /* Time to send another SGU (60)          */
    t41.value   60
} TUPGrpCfg;
```

TUPInitMgmtAPI

Initializes TUP management and opens a channel to the TX board.

Prototype

S16 **TUPInitMgmtAPI** (U8 ***board***, U8 ***srcent***)

Argument	Description
board	TX board number.
srcent	Calling application entity ID.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.

Details

Call **TUPInitMgmtAPI** before any other actions are taken.

TUPInitNSapCfg

Builds a basic configuration buffer that can be passed to **TUPSetNSapCfg**.

Prototype

void **TUPInitNSapCfg** (TUPNSapCfg ***cfg***, S16 ***spId***, S16 ***mtpSpId***, U32 ***opc***)

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td><b><i>cfg</i></b></td><td>Pointer to the configuration structure to fill:
<pre>
typedef struct _TUPNSapCfg
{
    spId   spId argument
    switchType   TUP_SW_ITU   /* TUP_SW_ITU or TUP_SW_CHN */
    ssf   SSF_NAT   /* SIO high nibble; one of:
                                       SSF_INTL,
                                       SSF_SPARE,
                                       SSF_NAT,
                                       SSF_RES   */
    origPointCode opc   /* local point code   */
    mtpSpId   mtpSpId argument /* MTP Sap to use, should
                                    * match MTP3 configuration */
    traceInd   0   /* trace on when nonzero   */
} TUPNSapCfg;
</pre></td></tr>
<tr><td><b><i>spId</i></b></td><td>Network service access point to configure.</td></tr>
<tr><td><b><i>mtpSpId</i></b></td><td>MTP NSAP to use that matches a network service access point in the MTP 3 configuration.</td></tr>
<tr><td><b><i>opc</i></b></td><td>Point code of this node at this service access point.</td></tr>
</table>

Return values

None.

Details

All unlisted elements in the TUPNSapCfg structure are set for the proper operation of the TUP stack and must not be changed.

Refer to *USAP and NSAP configuration* on page 60 for more information.

TUPInitUSapCfg

Builds a basic configuration buffer that can be passed to **TUPSetUSapCfg**.

Prototype

void **TUPInitUSapCfg** (TUPUSapCfg ***cfg**)

Argument	Description

cfg	Pointer to the configuration structure to fill:
	`typedef struct _TUPUSapCfg`
	`{`
	`  S16 switchType   TUP_SW_ITU /* TUP_SW_ITU or TUP_SW_CHN        */`
	`  S16 QCongOnsetLvl1 32       /* cfg onset value for level 0     */`
	`  S16 QCongAbateLvl1 16       /* cfg abatement value for level 1 */`
	`  S16 QCongOnsetLvl2 64       /* cfg onset value for level 1     */`
	`  S16 QCongAbateLvl2 48       /* cfg abatement value for level 2 */`
	`  S16 QCongOnsetLvl3 96       /* cfg onset value for level 2     */`
	`  S16 QCongAbateLvl3 80       /* cfg abatement value for level 3 */`
	`} TUPUSapCfg;`

Return values

None.

Details

All unlisted elements in the structure are set for the proper operation of the TUP stack and must not be changed.

Refer to *USAP and NSAP configuration* on page 60 for more information.

TUPQuietReset

Sends a request to the board to reset all circuits without generating TUP traffic.

Prototype

S16 **TUPQuietReset** (U8 ***board***)

Argument	Description
board	TX board number to configure.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

Details

Predefined groups are also reset. Any transitory groups are deleted.

Note: Use this call with extreme caution due to the possibility that the state of the circuits may not be identical between the network and NMS TUP after execution.

TUPResetCircuit

Sends a request to reset the given, pre-defined circuit.

Prototype

S16 **TUPResetCircuit** (U8 ***board***, CirIdx ***circID***)

Argument	Description
board	TX board number.
circID	Circuit index to reset.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	
TUP_DRIVER	Error occurred accessing the driver.
	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

TUPResetGroup

Sends a request to reset the given group of circuits.

Prototype

S16 **TUPResetGroup** (U8 ***board***, U16 ***groupID***)

	Description
board	
groupID	Group ID to reset.

Return values

Return value	Description
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

TUPSetCircCfg

Sets circuit configurations to the TX board. For more information, refer to *Circuits and groups configuration* on page 60.

Prototype

S16 **TUPSetCircCfg** (U8 ***board***, TUPCircCfg ***cfg**, S16 ***spId***)

Argument	Description
board	TX board number.
cfg	Pointer to the circuit configuration structure to fill. Refer to **TUPInitCircCfg**.
spId	NSAP for which to configure this circuit set.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

TUPSetGenCfg

Sends a basic configuration buffer to the TX board. Refer to *Configuration* on page 60 for more information.

Prototype

S16 **TUPSetGenCfg** (U8 ***board***, TUPGenParms ***cfg***)

Argument	Description
board	TX board number.
cfg	Pointer to the configuration structure to fill. Refer to **TUPInitGenCfg**.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

TUPSetNSapCfg

Sends a basic network service access point configuration buffer to the TX board. Refer to *USAP and NSAP configuration* on page 60 for more information.

Prototype

S16 **TUPSetNSapCfg** (U8 ***board***, TUPNSapCfg ****cfg***)

Argument	Description
board	TX board number.
cfg	Pointer to the network service access point structure. Refer to **TUPInitNSapCfg**.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

TUPSetUSapCfg

Sends a basic user service access point configuration buffer to the TX board. Refer to *USAP and NSAP configuration* on page 60 for more information.

Prototype

S16 **TUPSetUSapCfg** (U8 ***board***, TUPUSapCfg ***cfg***, U16 ***sapID***)

Argument	Description
board	TX board number.
cfg	Pointer to the USAP structure. Refer to **TUPInitUSapCfg**.
sapID	User service access point to configure.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

TUPTermMgmtAPI

Closes TUP management and the channel to the TX board.

Prototype

S16 **TUPTermMgmtAPI** (U8 ***board***)

Argument	Description
board	TX board number.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

Details

Call **TUPTermMgmtAPI** to close TUP management when the application has finished using it.

TUPTraceControl

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

Prototype

S16 **TUPTraceControl** (U8 ***board***, Bool ***onOff***, U32 ***flags***)

Argument	Description
board	TX board number.
onOff	Enable or disable tracing based on bitmap set by ***flags***.
flags	Bitmap of a particular trace that needs to be enabled or disabled.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

Details

The following TUP traces are available through the *tupmgr* command line option: EVENT, DATA, ERROR, WARNING, ELEMENT, and TIMER.

TUPUnblockCircuit

Sends a request to unblock the given circuit.

Prototype

S16 **TUPUnblockCircuit** (U8 ***board***, CirIdx ***circID***)

Argument	Description
board	TX board number.
circID	Circuit index to unblock.

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

TUPUnblockGroup

Sends a request to unblock the given group of circuits.

Prototype

S16 **TUPUnblockGroup** (U8 ***board***, U16 ***groupID***, U8 ***unblockTyp***)

Argument	Description
board	TX board number.
groupID	Group ID to unblock.
unblockTyp	Type of unblock. Possible values are: TUP_MUNBLOCK TUP_HUNBLOCK TUP_SUNBLOCK

Return values

Return value	Description
TUP_SUCCESS	
TUP_BOARD	***board*** is out of range.
TUP_DRIVER	Error occurred accessing the driver.
TUP_FAILED	Task on the TX board reported a failure.
TUP_UNBOUND	Application failed to call **TUPInitMgmtAPI** prior to this call.

8 Demonstration programs and utilities

Summary of the demonstration programs and utilities

NMS TUP provides the following demonstration programs and utilities:

Program	Description
tupcfg	Downloads the TUP configuration to the TX board at boot time.
tupmgr	Monitors and manages the status of the TUP layer.
tupterm	Demonstrates how the TUP service accepts an incoming call from the specified TX board.
tuporig	Demonstrates how the TUP service generates an outbound call to the specified TX board.

TUP configuration utility: tupcfg

Scans the TUP configuration text file and downloads the configuration to the TUP task on the TX board at boot time.

Requirements

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

Procedure

Follow this procedure to run *tupcfg*:

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>From the command line prompt, navigate to the <i>\nms\tx\samples\tup\tupcfg\</i> directory under Windows or the <i>/usr/bin/</i> directory under UNIX.</td></tr>
<tr><td>2</td><td>Enter the following command:

```
tupcfg options
```

where <b><i>options</i></b> include:
<table>
<tr><th>Options</th><th>Description</th></tr>
<tr><td>-b <b><i>board</i></b></td><td>TX board number to which the TUP configuration is downloaded. Default = 1.</td></tr>
<tr><td>-f <b><i>filename</i></b></td><td>Name and location of the TUP configuration file to be downloaded.</td></tr>
</table>
The TUP configuration program scans the information in the ASCII file (specified with the -f option) and downloads this information to the task on the TX board.</td></tr>
</table>

Details

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

TUP layer status: tupmgr

Monitors the status of the TUP layer after the TUP configuration is downloaded to the TX board with *tupcfg*. The TUP manager (*tupmgr*) provides a command line interface that enables an application to set alarm levels, trace buffers, and view TUP statistics.

Requirements

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

Procedure

Follow this procedure to run *tupmgr*:

Step	Action
1	From the command line prompt, navigate to the \nms\tx\samples\tup\tupmgr\\ directory under Windows or the */usr/bin/* directory under UNIX.
2	Enter the following command: `tupmgr -b board` where ***board*** is the TX board on which the TUP layer is loaded.

The *tupmgr* program supports the following commands:

Command	Description
VALIDATE ***circuit***	Validates the specified ***circuit***.
BLOCK ***circuit***	Blocks TUP ***circuit***.
UNBLOCK ***circuit***	Unblocks TUP ***circuit***.
RESET ***entity number***	Resets the specified ***entity*** (CIRCUIT or GROUP) ***number***.
MBLOCK ***circuit***	Maintenance blocks the specified ***circuit*** group.
HBLOCK ***circuit***	Hardware blocks the specified ***circuit*** group.
SBLOCK ***circuit***	Software blocks the specified ***circuit*** group.
MUNBLOCK ***circuit***	Maintenance unblocks the specified ***circuit*** group.
HUNBLOCK ***circuit***	Hardware unblocks the specified ***circuit*** group.
SUNBLOCK ***circuit***	Software unblocks the specified ***circuit*** group.
DELETE ***entity number***	Deletes the specified ***entity*** (CIRCUIT or GROUP) ***number***. If CIRCUIT is selected, the ***count*** of circuits to delete must also be specified.
STATUS ***entity number***	Retrieves status of current ***entity*** (CIRCUIT or GROUP) ***number***.
STATS ***entity number***	Retrieves statistics for the specified ***entity*** (NSAP, GROUP, or CIRCUIT) ***number***.
GET ***entity number***	Retrieves configuration for the specified ***entity*** (GEN, USAP, NSAP, GROUP, CIRCUIT) ***number***.
TRACE ***entity ON | OFF***	Turns buffer tracing ON or OFF for the specified ***entity*** (EVENT, DATA, ERROR, WARNING, ELEMENT, or TIMER).
BOARD ***board***	Switches to a new target ***board***.
Q	Quits the application.

Details

The TUP manager program is available in both source code and executable formats. The source code demonstrates how to use TUP management to integrate SS7 TUP layer management into custom configuration management systems.

Accepting incoming calls: tupterm

Demonstrates how the TUP service accepts an incoming phone call from the specified TX board.

Requirements

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

Procedure

Follow this procedure to run *tupterm*:

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>From the command line prompt, navigate to the <i>\nms\tx\samples\tup\</i> directory under Windows or the <i>/usr/bin/</i> directory under UNIX.</td></tr>
<tr><td>2</td><td>Enter the following command:
<code>tupterm <b><i>options</i></b></code>
where <b><i>options</i></b> include:
<table><tr><th>Option</th><th>Description</th></tr><tr><td>-b <b><i>board</i></b></td><td>TX board number on which to bind. Default = 1.</td></tr></table></td></tr>
<tr><td>3</td><td>Press any key to exit the program.</td></tr>
</table>

Generating outbound calls: tuporig

Demonstrates how the TUP service generates an outbound call to the specified TX board.

Requirements

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

Procedure

To run *tuporig*:

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>From the command line prompt, navigate to the <i>\nms\tx\samples\tup\</i> directory under Windows or the <i>/usr/bin/</i> directory under UNIX.</td></tr>
<tr><td>2</td><td>Enter the following command:
<code>tuporig <b><i>options</i></b></code>
where <b><i>options</i></b> include:
<table>
<tr><th>Option</th><th>Description</th></tr>
<tr><td>-b <b><i>board</i></b></td><td>TX board number on which to bind. Default = 1.</td></tr>
<tr><td><b><i>called number</i></b></td><td>Phone number to which the call is placed.</td></tr>
<tr><td><b><i>circuit index</i></b></td><td>Circuit index on which the call is placed.</td></tr>
<tr><td><b><i>calling number</i></b></td><td>Phone number from which the call is placed.</td></tr>
</table>
Refer to the source code for more options.</td></tr>
<tr><td>3</td><td>Press <b>r</b> to release the call.</td></tr>
<tr><td>4</td><td>Press <b>q</b> to quit the program.</td></tr>
</table>

9 Information elements and events reference

Usage information overview

This section provides information on the NMS TUP:

- Information elements (IEs)
- Events

Data structures

C language definitions for all information element structures, event structures, and associated constants are provided in the *tupmsgs.h* file.

Applications must code information elements not relevant to their application as NOT_PRESENT.

Coding of presence indicators

Each information element within an event contains a presence indicator to specify whether or not to include it in an outgoing event, or whether or not it was received in an incoming event.

Presence indicators are coded with the following values (definitions included in *tupmsgs.h*):

```
#define NOT_PRESENT  0   /* field not present in incoming msg
                          * or not to be populated in outgoing msg  */
#define PRESENT      1   /* field is present in incoming msg or
                          * should be included in outgoing msg      */
```

Information elements (IEs)

This topic specifies the layout of the following information elements (IEs) that are contained in the events passed between the application and NMS TUP:

- Element header
- Token string
- Address signals
- Answer type information
- Automatic congestion level
- Calling party category
- Charge information
- Closed user group interlock code
- Initial address indicator
- LineID
- Message indicator
- Point code
- Range information
- Response indicator
- Trunk ID

Element header

Each information element contains an element header as the first field in the structure. The element header consists of the presence indicator for the entire information element.

```
typedef struct tupelmtHdr  /* element header   */
{
    Bool pres;             /* present          */
    U8 spare1;             /* for alignment    */
    U16 spare2;            /* for alignment    */
} TupElmtHdr;
```

Token string

Consists of the presence indicator for the entire token string, a length field, and the string itself.

```
typedef struct tuptknStr   /* token string         */
{
    U8 spare0;             /* present flag         */
    U8 len;                /* length               */
    U16 spare1;            /* for alignment        */
    U8 val[(TUP_MF_SIZE_TKNSTR + 4) & 0xffc];
                           /* string               */
} TupTknStr;
```

Address signals

Contains the address signals of a called number.

```
typedef struct addrSignals
{
    TupElmtHdr eh;         /* element header */
    TupTknStr addrSigs;
} AddrSigs;
```

The addrSigs field is coded the same as in the LineID information element.

Answer type information

Contains the type of answer message to send or the type received.

```
typedef struct ansType
{
    TupElmtHdr eh;    /* element header  */
    U8 answerType;
    U8 spare2;
    U16 spare1;
} AnswerType;
```

answerType is coded to one of the following:

Hex	Message
0x0	ANU_NO_CHARGE
0x1	ANU_CHARGE
0x2	ANU_UNQUALIFIED

Automatic congestion level

Indicates that a particular level of congestion exists at the sending exchange.

```
typedef struct autoCongLvl
{
    TupElmtHdr eh;     /* element header     */
    U8 auCongLvl;      /* congestion level   */
    U8 spare2;
    U16 spare1;
}TupAutoCongLvl;
```

The auCongLvl field is coded to one of the following values:

Hex	Message
0x0	TUPACLVL_LVL1
0x1	TUPACLVL_LVL2

Calling party category

Contains the category of the calling party.

```
typedef struct callPartyCategory
{
    TupElmtHdr eh;       /* element header */
    U8 callPartyCat;
    U8 spare2;
    U16 spare1;
} CallPartyCat;
```

The callPartyCat field is coded to one of the following values:

Hex	**Message**
0x0	UNKNOWN_SOURCE
0x1	OPERATOR_FRENCH
0x2	OPERATOR_ENGLISH
0x3	OPERATOR_GERMAN
0x4	OPERATOR_RUSSIAN
0x5	OPERATOR_SPANISH
0x6	MUT_AGREE_CHINESE
0x7	MUT_AGREEMENT
0x8	MUT_AGREE_JAPANESE
0x9	NATIONAL_OPERATOR
0x0a	ORDINARY_TOLL_CALLER
0x0b	PRIORITY_TOLL_CALLER
0x0c	DATA_CALL
0x1d	TEST_CALL
0xf	PAY_PHONE (not supported in China)
0x10	ORDINARY_NO_CHG (China only)
0x11	ORDINARY_PERIODIC (China only)
0x12	ORDINARY_METER_IMMED (China only)
0x13	ORDINARY_PRINTER_IMMED (China only)
0x14	PRIORITY_NO_CHG (China only)
0x15	PRIORITY_PERIODIC (China only)
0x18	ORDINARY_LOCAL (China only)

Charge information

Contains a raw 16-bit piece of charging information.

```
typedef struct chgInfo
{
    TupElmtHdr eh;    /* element header  */
    U16 info;
    U16 spare1;
} ChargeInfo;
```

Closed user group interlock code

Identifies a closed user group within a network.

```
typedef struct clsedUserGrp
{
    TupElmtHdr eh;       /* element header   */
    U8 grpInd;           /* group indicator  */
    U8 spare2;
    U16 spare1;
    U32 interlockCode;   /* interlock Code   */
} ClosedUserGroup;
```

The fields are encoded as follows:

Field	Description
interlockCode	32-bit binary code assigned by the network administrator.
grpInd	Set to one of the following values: 0x00 = CUG_ORDINARY_CALL 0x01 = CUG_SUCCESSFUL_CHECK 0x02 = CUG_ACCESS_ALLOWED 0x03 = CUG_ACCESS_NOT_ALLOWED

Initial address indicator

Contains an indicator of the caller:

```
typedef struct iamMessageInd
{
    TupElmtHdr eh;      /* element header  */
    U16 msgInd;         /* indicator       */
    U16 spare1;
} IAMMessageInd;
```

The msgInd field is coded by using OR to combine one or more of the following values:

Hex	Message
0x000	IAM_MI_LOCAL_SUBSCRIBER
0x002	IAM_MI_NATIONAL
0x003	IAM_MI_INTERNATIONAL
0x000	IAM_NO_SATELLITE
0x004	IAM_ONE_SATELLITE
0x000	IAM_MI_CONT_CHECK_NOT_REQ
0x010	IAM_MI_CONT_CHECK_REQ
0x020	IAM_MI_CONT_CHECK_PERFORMED_PREVIOUS
0x040	IAM_MI_OUT_ECHO_SUPP_INCLUDED
0x080	IAM_MI_IN_CALL_INTERNATIONAL
0x100	IAM_MI_REDIRECTED_CALL
0x200	IAM_MI_DIGITAL_PATH_REQUIRED
0x400	IAM_MI_ALL_SS7_PATH

LineID

Contains the information necessary to identify the calling or called party.

```
typedef struct lineID   /* Called Party Number    */
{
    TupElmtHdr eh;      /* element header         */
    U8 addrInd;         /* nature of indicator    */
    U8 spare2;
    U16 spare1;
    TknStr addrSigs;    /* Address Signal         */
} LineID;
```

The fields in the LineID structure are encoded as follows:

Field	Description
addrInd	0x00 = TUPSUBSNUM Subscriber number 0x02 = TUPNATNUM Nationally significant number 0x03 = TUPINTNATNUM International number 0x00 = TUPNOTRES Number not restricted 0x04 = TUPRESTRICTED Number is restricted 0x00 = TUPNOTINC No indic incomplete number 0x08 = TUPINCOMPLETE Number incomplete
addrSigs	Actual address digits.

addrSigs is encoded as follows:

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

Each digit is encoded with the following bit pattern:

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

TUPASCIItoADDR and **TUPADDRtoASCII** convert between ASCII format and this format.

Message indicator

Contains an address complete indicator or extended unsuccessful backward message (EUM) subscriber busy.

```
typedef struct messageInd
{
    TupElmtHdr eh;    /* element header  */
    U8 indicator;
    U8 spare2;
    U16 spare1;
} msgInd;
```

The indicator field for an ACM is coded by using OR to combine one or more of the following:

Hex	Message
0x00	ACM_COMPLETE
0x01	ACM_COMPLETE_CHARGE
0x02	ACM_COMPLETE_NO_CHARGE
0x03	ACM_COMPLETE_COINBOX
0x04	ACM_SUBSCRIBER_FREE
0x08	ACM_INCOMING_ECHO_SUPPRESSED
0x10	ACM_CALL_FORWARDED (not used)
0x20	ACM_ALL_SS7_PATH (not used)

The indicator field for an EUM is coded with the following:

Hex	Message
0x01	EUM_SUBSCRIBER_BUSY

Point code

Identifies a point code.

```
typedef struct pntCode
{
    TupElmtHdr eh;     /* element header     */
    U32 pointCode;     /* Point Code         */
} PointCode;
```

Format of the pointCode field is in hexadecimal notation.

For 24-bit point codes, 0xABCDEF properly sets the 3 LSB of the field.

For 14-bit point codes, only the LSB and the six least significant bits of the next LSB are used.

Range information

Pertains to the number of circuits for which a group message has relevant information held in an 8-bit raw value:

```
typedef struct grmRange
{
    TupElmtHdr eh;        /* element header       */
    U8 range;
    U8 spare2;
    U16 spare1;
    U8 statusMap[256];    /* set if range != 0    */
} Range;
```

The range field is defined to be 0 (zero) for predefined groups. If the group is dynamic, the value must be one less than the number of circuits in the group. For example, if the group is 10 circuits, the range is 9. The statusMap bytes must be set to 0 (zero) or 1, depending on whether or not the action is taken on that circuit in the group. For 10 circuits, if you do not want to reset the eighth one, set all bytes to 1 except for the statusMap[7].

Response indicator

Contains a request or response indicator.

```
typedef struct respInd
{
    TupElmtHdr eh;         /* element header  */
    U8 indicator;
    U8 spare2;
    U16 spare1;
} RespInd;
```

The indicator field is calculated using OR to combine one or more of the following values. There are separate values for the information request (GRQ) and information response (GSM):

Hex	Message
0x01	GSM_CALLING_PARTY_CATEGORY
0x02	GSM_CALLING_LINE_ID
0x04	GSM_INCOMING_EXCH_ID
0x08	GSM_ORIG_CALLED_ADDRESS
0x10	GSM_OUTGOING_ECHO_SUPPRESSOR
0x20	GSM_MALICIOUS_CALL_ID
0x40	GSM_HOLD_INDICATOR
0x01	GRQ_CALLING_PARTY_CATEGORY
0x02	GRQ_CALLING_LINE_ID
0x04	GRQ_ORIG_CALLED_ADDRESS
0x08	GRQ_MALICIOUS_CALL_ID
0x10	GRQ_HOLD_INDICATOR
0x20	GRQ_OUTGOING_ECHO_SUPPRESSOR

Trunk ID

Contains the trunk ID for a call.

```
typedef struct trunkid
{
    TupElmtHdr eh;     /* element header        */
    U8 idType;         /* type of exch ID       */
    U8 spare2;
    U16 spare1;
    TknStr exchID;     /* exch ID               */
    TknStr trunkID;    /* trunk ID-not used     */
} TrunkID;
```

Set idType to one of the following:

Hex	Value
0x01	GSM_ID_TYPE_IND_POINT_CODE
0x02	GSM_ID_TYPE_IND_AVAIL_LINE_ID

The exchID field is coded with the digit values listed in the LineID information element or a point code, depending on the value of idType.

The trunkID field is not used.

Events

This topic specifies the layout of the events passed between the application and NMS TUP:

- Connect request
- Connect response
- Connect status
- Status
- Release
- All events

Connect request

```
typedef struct tupConEvnt          /* TUP Connect Event  */
{
    CallPartyCat     cgPtyCat;     /* IAM, IAI           */
    IAMMessageInd    msgInd;       /* IAM, IAI           */
    AddrSigs         addrSigs;     /* IAM, IAI           */
    LineID           cgPtyNum;     /* IAI                */
    LineID           origCdNum;    /* IAI                */
    ClosedUserGroup  clUsrGrp;     /* IAI                */
} TupConEvnt;
```

Connect response

```
typedef struct tupConRspEvnt
{
    AnswerType       answerType;    /* ANC, ANN, ANU  */
} TupConRspEvnt;
```

Connect status

```
typedef struct  tupCnStEvnt
{
    RespInd respInd;          /* GRQ, GSM   */
    MsgInd msgInd             /* ACM, ACC   */
    TupAutoCongLvl congLvl;   /* ACC        */
    AddrSigs addrSigs;        /* SAM, SAO   */
    CallPartyCat cgPtyCat;    /* GSM        */
    LineID cgPtyNum;          /* GSM        */
    LineID origCdNum;         /* GSM        */
    TrunkID trunkID;          /* GSM        */
    ChargeInfo chargeInfo;    /* MPM        */
} TupCnStEvnt;
```

Status

```
typedef struct tupStaEvnt   /* Status Event   */
{
    MsgInd msgInd;          /* EUM            */
    PointCode pointCode;    /* EUM            */
    Range  range;           /* all GRM's      */
} TupStaEvnt;
```

Release

```
typedef struct tupRelEvnt  /* Release Event    */
{
    Cause cause;           /* NOT USED         */
} TupRelEvnt;
```

All events

This structure is filled if there is an error to report. It is included in the TupAllSdus structure. The evntType field of the information block is filled with ERRORIND in the error report case. Refer to **TUPRetrieveMessage**.

```
typedef struct tupErrInfo
{
    U8 origIndType;    /* original indicator       */
    U8 origEvntType;   /* original event type      */
    U32 errorCode;     /* error code - see below   */
} TupErrInfo;
```

errorCode is filled with one of the following values:

Inex	Value	Description
1	TUP_NOMEM	No memory
2	TUP_INVNSAP	Invalid NSAP
3	TUP_INVUSAP	Invalid USAP
4	TUP_INVGRP	Invalid group number
5	TUP_INVCIRC	Invalid circuit index
6	TUP_SAPEXIST	SAP exists already
7	TUP_CIRCEXIST	Circuit exists already
8	TUP_MTPERR	Error received from MTP
9	TUP_INVSTATE	Circuit is in invalid state for this action
10	TUP_UNKEVT	Unknown event
11	TUP_INVENT	Invalid entity
12	TUP_INVPKT	Invalid packet received
13	TUP_INVINST	Invalid instance
14	TUP_BLOCKED	Circuit is blocked
15	TUP_INVSWITCH	Invalid switch type

This union is the returned structure from **TUPRetrieveMessage** in the event parameter:

```
typedef struct tupAllSdus        /* all SDU messages */
{
    union
    {
      TupConEvnt conEvnt;
      TupConRspEvnt conRspEvnt;
      TupCnStEvnt conStEvnt;
      TupRelEvnt relEvnt;
      TupStaEvnt staEvnt;
    } m;
    TupErrInfo errInfo;
} TupAllSdus;
```

Index