Dialogic® NaturalAccess™ ISUP Layer Developer's Reference Manual

July 2009 64-0453-01

www.dialogic.com

Copyright and legal notices

Revision history

Revision	Release date	Notes
9000-6471- 30	August 2001	GJG, SS7 3.8 Beta
9000-6471-31	February 2002	MVH, SS7 3.8
9000-6471-32	November 2003	SRG, SS7 4.0 Beta
9000-6471-33	April 2004	MCM, SS7 4.0
9000-6471-34	April 2005	SRG, SS7 4.2
9000-6471-35	July 2008	DEH, SS7 5.0 Beta
9000-6471-36	September 2008	DEH/LBG, SS7 5.0
64-0453-01	July 2009	LBG, SS7 5.1
Last modified: July 7, 2009		

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

Table Of Contents

1 Introduction

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

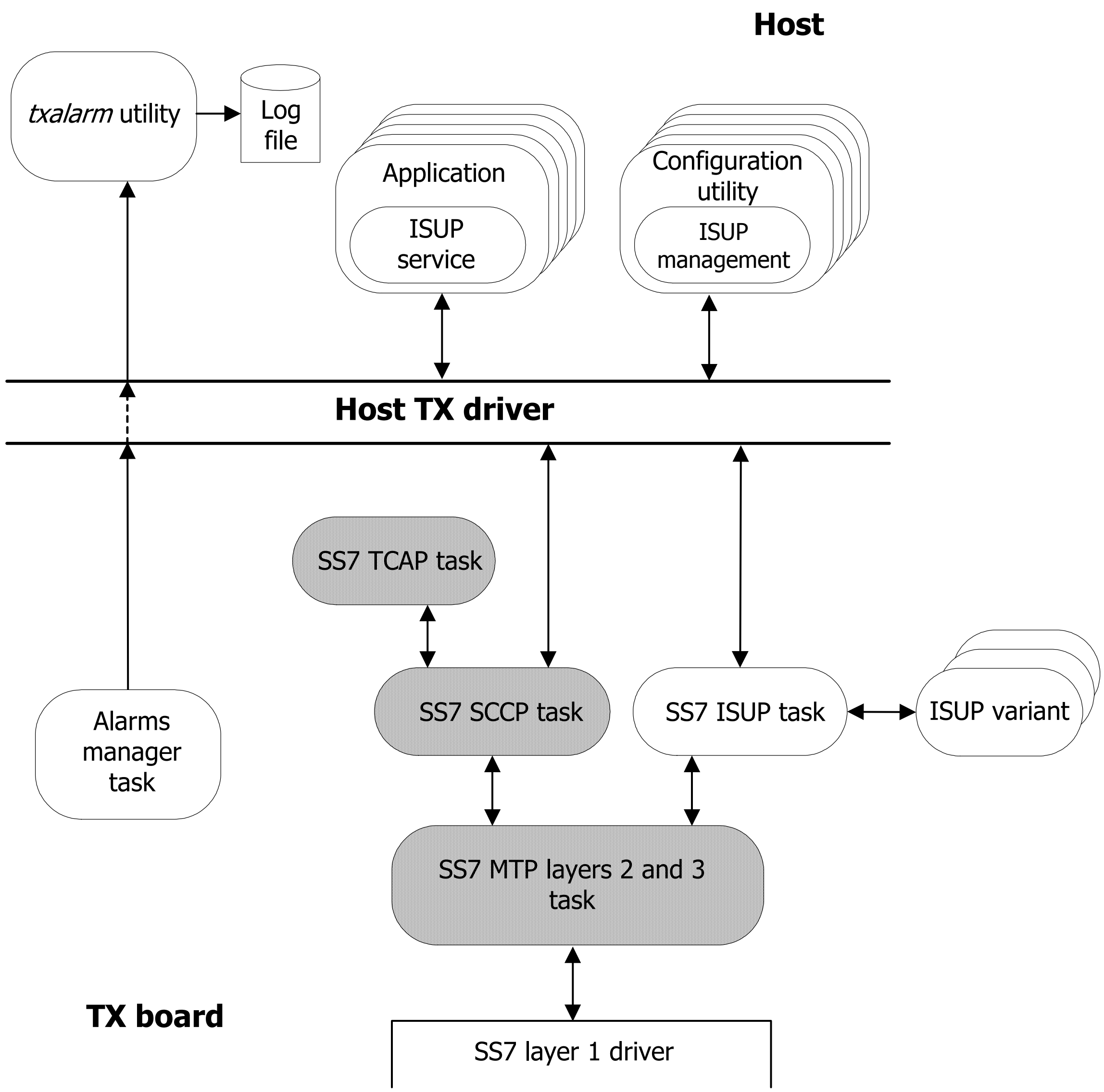
Note: The product to which this document pertains is part of the NMS Communications Platforms business that was sold by NMS Communications Corporation (“NMS”) to Dialogic Corporation (“Dialogic”) on December 8, 2008. Accordingly, certain terminology relating to the product has been changed. Below is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

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

2 SS7 overview

SS7 architecture

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



The TX board consists of the following components:

- ISUP task that implements the SS7 ISUP layer.
- ISUP variant task that contains the encoding and decoding messages tables for a specific ISUP variant (for example, ITU White or ANSI 95).
- 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 TCAP task that implements the SS7 TCAP layer.
- TX alarms manager task that collects unsolicited alarms (status changes) generated by the SS7 tasks and forwards them to the host for application-specific alarm processing.

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

ISUP task

The ISUP 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), direction, destination point code, and routing instructions are specified in the ISUP configuration file. The ISUP task reads the ISUP configuration file at startup.

For outgoing call setup requests, the ISUP task does not select a circuit for the connection. The application must specify the circuit to be connected.

For incoming calls, the ISUP task verifies that the circuit state and characteristics, such as bearer capability, are compatible with the incoming call request parameters before passing the incoming call indication to the application.

For both incoming and outgoing calls, the ISUP task provides all necessary connection timers. The ISUP task notifies both the application and the far exchange with necessary indications, such as connection clearing, when critical timers expire.

The ISUP task:

- Provides circuit supervision for the duration of the connection.
- Adjusts the circuit state as needed based on requests from the application and ISUP 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.

Bearer Independent Call Control extensions

In support of IP network based signaling, the Bearer Independent Call Control (BICC) stack capability is configured as a variant of ISUP. This capability allows network operators to offer the complete set of PSTN/ISDN services, including all supplementary services, over a variety of packet networks.

Support for BICC also allows network operators to gradually migrate their PSTN/ISDN networks to high-capacity packet networks. This is a vital step in the evolution toward integrated multi-service platforms, which can offer both voice and data services that are IP enabled. BICC can be transported over SS7 MTP3 or SIGTRAN (M3UA/SCTP).

The NaturalAccess™ BICC implementation supports Capability Sets 1 and 2 (CS1 and CS2) features such as basic call setup, forward or backward bearer setup, and supplementary services as specified in the Q.1901 and Q.1902.x specifications.

3 ISUP programming model

Programming model overview

NaturalAccess™ ISUP supports one or more applications with service access points (SAPs). One SAP is defined for each application that uses the ISUP service. An application binds to a particular service access point at initialization and specifies the service access point ID to which it wants to bind.

A switch type (CCITT, ANSI-88, or ANSI-92) is associated with each user service access point in the ISUP configuration file. The ISUP task associates the switch type with its configured network connections. Multiple service access points, and hence multiple applications, for a particular switch type can be defined for outgoing call requests.

Incoming messages that are associated with a previously issued outgoing request are always routed to the service access point that issued the original outgoing request. Incoming call requests, however, are always routed to the first service access point with a switch type that matches the switch type of the network connection over which the request was received.

Note: The ISUP configuration file specifies the number of service access points and the characteristics of each one. Refer to the *Dialogic® NaturalAccess™ Signaling Software Configuration Manual* for more information.

Entity and instance IDs

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

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

Instance IDs identify the processor on which the entity executes. The host is always processor 0 (zero). Therefore, all host-resident ISUP 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.

ISUP service and management functions

NaturalAccess™ ISUP provides service functions and management functions.

Service functions

The ISUP service functions provide the application access to the ISUP layer services. Applications invoke ISUP services by calling ISUP request functions that send an ISUP message to a remote exchange or endpoint. Request function parameters are converted to messages. The host operating system TX driver sends these parameters to the ISUP task.

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

All ISUP service functions are asynchronous. Completion of the function implies only that the function was successfully initiated (a request message was queued to the ISUP task). Errors detected by the ISUP 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 ISUP layer, are passed to application processes as asynchronous events. All events for a particular user service access point (subsystem) 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 ISUP raw format to API format.

For more information, refer to the Using the ISUP service section and the *ISUP service function summary* on page 41.

Management functions

Unlike the ISUP service functions that send and receive messages asynchronously, each ISUP management function generates a request followed immediately by a response from the TX board. ISUP management functions block the calling application waiting for this response for a maximum of five seconds, but typically a few hundred milliseconds. The management functions return an indication as to whether or not an action completed successfully.

For this reason, ISUP management functions are typically used by one or more management applications. Separate applications use the ISUP service functions. ISUP management is packaged as a separate library with its own interface header files.

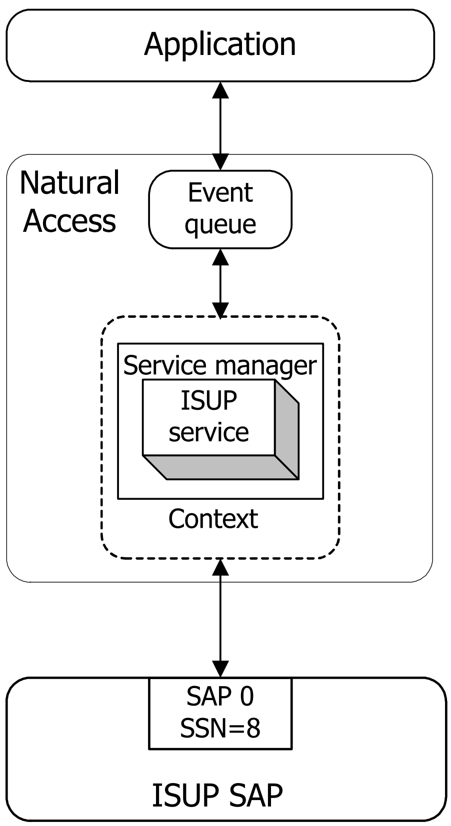
For more information, refer to the *ISUP management function summary* on page 131.

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 an ISUP service access point is a unique Natural Access context. Contexts are created with **ctaCreateContext**.

All events and messages from the ISUP 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.

ISUP supports a single-context, single-queue application programming model, as shown in the following illustration:

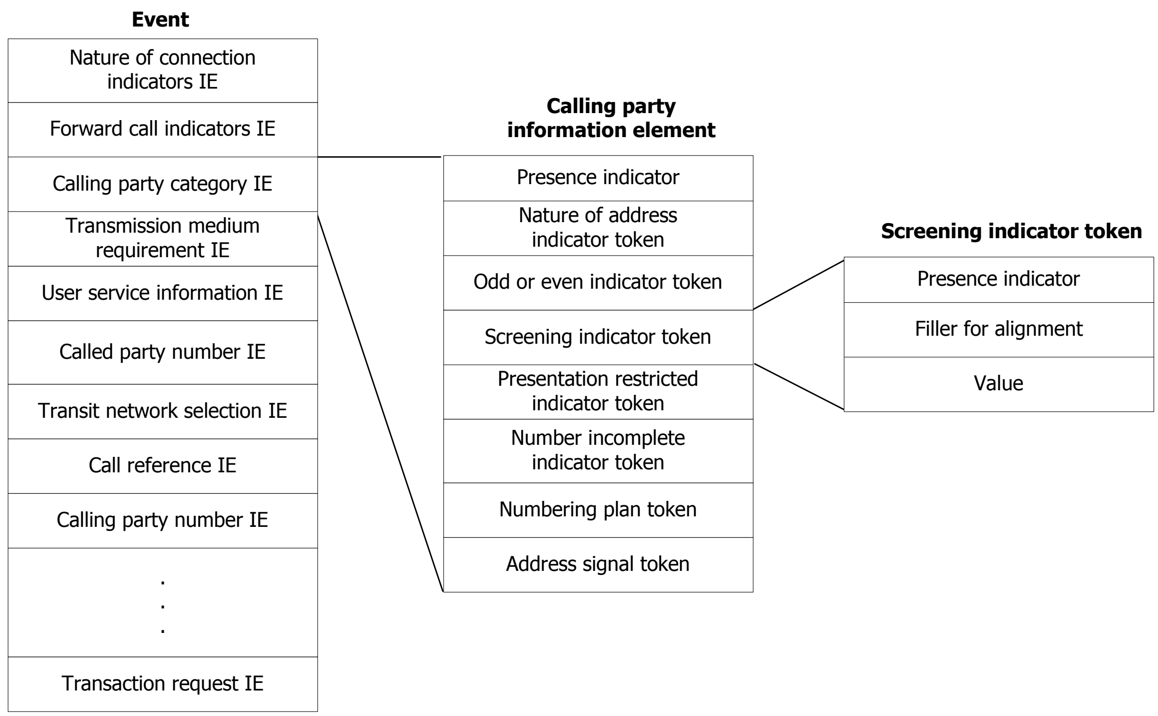


Signaling parameters

Signaling parameters are passed between the application and the ISUP task in the form of events. The following table provides a description of the signaling parameter components:

Component	Description
Events	Fixed format structures consisting of one or more information elements (IE).
Information elements	Fixed format structures consisting of a flag indicating their presence or absence from the corresponding ISUP message, and one or more tokens, or fields.
Token	Structure consisting of a flag indicating its presence or absence, possible filler to ensure proper byte alignment, and a value.

The following structure simplifies applications by enabling them to operate on fixed format structures rather than the variable length and variable formats employed by the ISUP protocol.



Global messaging toolkit

The global messaging toolkit (GMT) is a set of ISUP features that enables applications to support ISUP variants without additional software changes from Dialogic.

Feature	Description
Raw messages	Enables an application to send or receive complete ISUP messages and to support new messages without waiting for software updates from Dialogic. Any message type can be sent as a raw message. If this feature is enabled, inbound messages with a message type that the stack is not aware of can be sent to the application as a raw message. The first byte in the RAW structure is the message type, as the stack prepends the routing label. A few next state parameters are available if one of the new messages replaces some of the basic messages in an ISUP call, for example, WAIT_ACM. For more information, refer to *ISUPRawReq* on page 54.
Extended elements	Appends additional optional elements on the end of existing ISUP messages. This is useful for variants that can add new information elements to existing ISUP messages, often the IAM. This feature must be enabled and works for inbound unknown information element types. It can also be used to send additional parameters. Messages that do not have optional parameters in the specification definition cannot add these extended elements. For example, the RSC does not contain any optional parameters, so this feature cannot be used with an RSC. For more information, refer to the *Information elements overview* on page 185.

Data tracing

Review the raw ISUP data packets when debugging applications. This feature, called data tracing, can be enabled or disabled using the *isupmgr* demonstration program, or it can be enabled or disabled in the ISUP configuration. Refer to the *Dialogic® NaturalAccess™ Signaling Software Configuration Manual* for more information.

When enabled, the raw data packets are sent to the *ss7trace* program that is included with the MTP3 product (including the *ss7trace* source code). Messages are presented as hexadecimal dumps with the source and destination pulled out for viewing. The first two bytes of each trace are the CIC for the message, and the third byte is the message type.

4 Using the ISUP service

Setting up the Natural Access environment

Before calling any ISUP service functions, the application must:

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

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

Initializing the Natural Access environment

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

```
CTA_INIT_PARMS      isupInitparms       = {0};
CTA_SERVICE_NAME    isupServiceNames[]  = {{"ISUP", "ISUPMGR"}};
...
isupInitparms.size              = sizeof(CTA_INIT_PARMS);
isupInitparms.traceflags        = CTA_TRACE_ENABLE;
isupInitparms.parmflags         = CTA_PARM_MGMT_SHARED;
isupInitparms.ctacompatlevel    = CTA_COMPATLEVEL;

Ret = ctaInitialize(isupServiceNames, 1, &isupInitparms);
if (Ret != SUCCESS) {
    printf("ERROR code 0x%08x initializing Natural 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, "IsupSAP-%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 ISUP service

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

Field	Description
board	TX board number.
srcInst	Calling application instance ID.
srcEnt	Calling application entity ID.
AutoBind	Determines if the application should bind immediately to the ISUP task. 1 = yes 0 = no
spId	ISUP service access point ID on which to bind.
suId	Calling application service user ID.
API queue size	Maximum number of requests that can be queued to the board in the ISUP service. Valid range is 128 to 1024. Default = 128.

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

```
CTA_SERVICE_DESC isupOpenSvcLst[num_isupServices] = {{{"ISUP", "ISUPMGR"}, {0}, {0}, {0}}
};

isupOpenSvcLst[0].svcargs.args[0] = boardNum; /* board number           */
isupOpenSvcLst[0].svcargs.args[1] = INST_ID;  /* srcInst                */
isupOpenSvcLst[0].svcargs.args[2] = ENT_ID;   /* dprChan                */
isupOpenSvcLst[0].svcargs.args[3] = 1;        /* AutoBind? (yes=1,no=0) */
isupOpenSvcLst[0].svcargs.args[4] = SAP_ID;   /* spId                   */
isupOpenSvcLst[0].svcargs.args[5] = SAP_ID;   /* suId                   */
isupOpenSvcLst[0].svcargs.args[6] = 1024;     /* API queue size         */
```

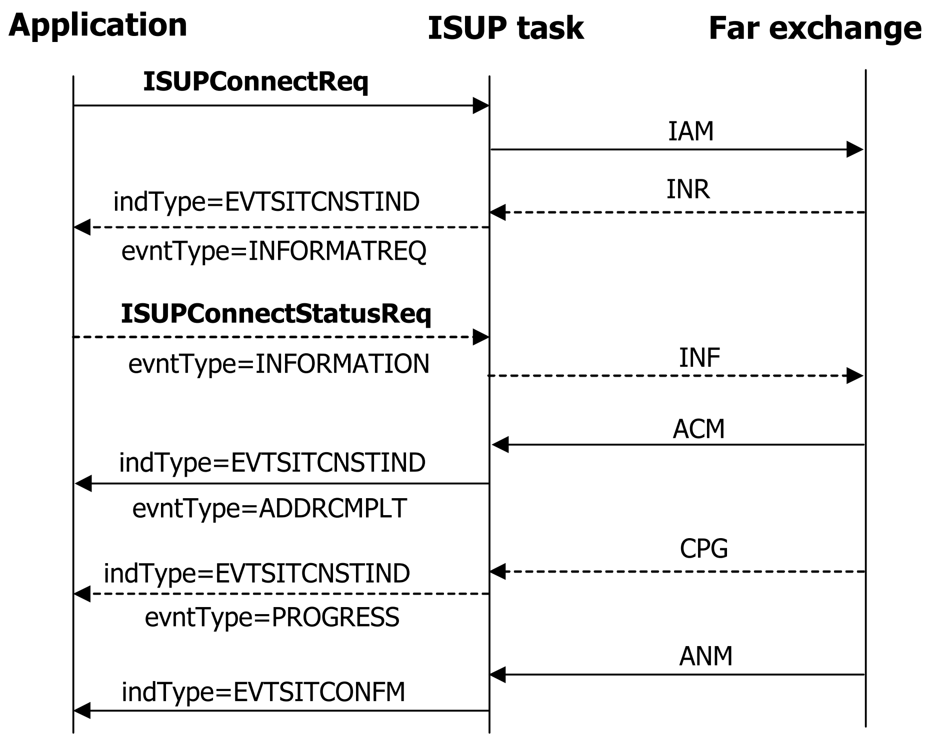
ctaOpenServices is an asynchronous function. The return from the function indicates that the bind operation initiated. Once completed, a CTAEVN_OPEN_SERVICES_DONE event is returned to the application:

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


Ret = ctaOpenServices( ctaHd, isupOpenSvcLst, 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 ISUP service [%s]\n", sErr );
    ctaDestroyQueue( pSap->ctaQueue );    /* Destroys context too */
    return( ... );
}
```

Establishing connections

The application initiates a circuit switched connection by invoking **ISUPConnectReq** resulting in the generation of an ISUP initial address message (IAM) to the far exchange. Alternatively, the far exchange can initiate the connection by sending the IAM message. The application receives a connect indication (EVTSITCONIND) event.

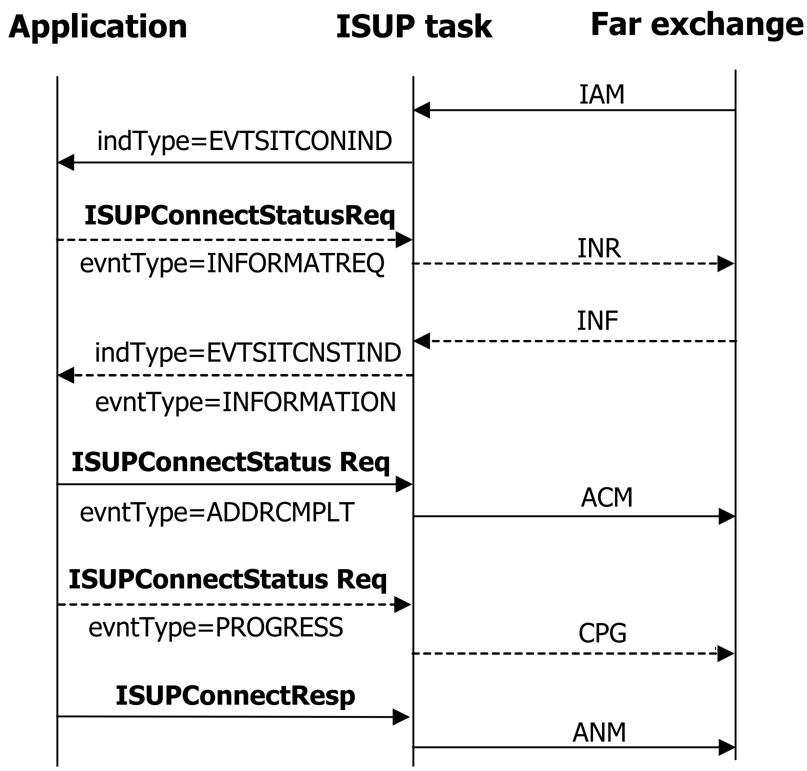
The following illustration shows the functions and events used to establish an outgoing connection:



Dashed lines indicate optional messages.

During the connection establishment phase, the application exchanges call progress and other status information with the far exchange by invoking **ISUPConnectStatusReq** with an event type and by receiving ISUP connect status indication (EVTSITCNSTIND) events from the ISUP task.

The following illustration shows the functions and events used to establish an incoming connection:



Dashed lines indicate optional messages.

The connection establishment phase ends when one of the following events occur:

- The application receives a connect confirmation (EVTSITCONCFM) event (far exchange sent answer or connect message)
- The application invokes **ISUPConnectResp** to signal to the far end that the connection is established for an incoming call.

Establishing connections with continuity check required

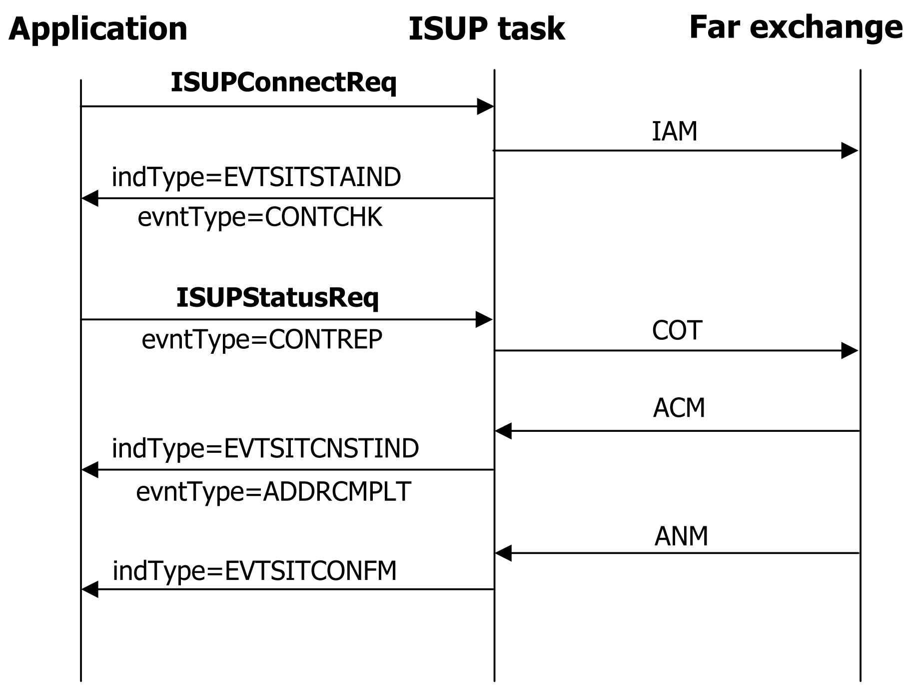
The application initiates a circuit switched connection with continuity check required by invoking **ISUPConnectReq** with the Nature of connection indicators IE field contChkInd set to CONTCHK_REQ. This results in the generation of an ISUP initial address message (IAM) to the far exchange.

When the initial address message has been sent, the ISUP layer issues a status indication (EVTSITSTAIND) with event type CONTCHK to the application. This is an indication to the application to perform the continuity check on the circuit.

The application then reports the results of the continuity check by calling **ISUPStatusReq** with event type of CONTREP. Assuming that the continuity test is successful, call processing resumes normally. If the continuity test fails, the application can request a recheck of the circuit as described in *Continuity recheck* on page 28. Optionally, the application can release the connection as described in *Clearing connections* on page 30.

Note: The BICC variant does not support establishing connections with continuity check required. To establish connections with BICC, see *Establishing connections* on page 24.

The following illustration shows the functions and events used to establish an outgoing connection with continuity check required:

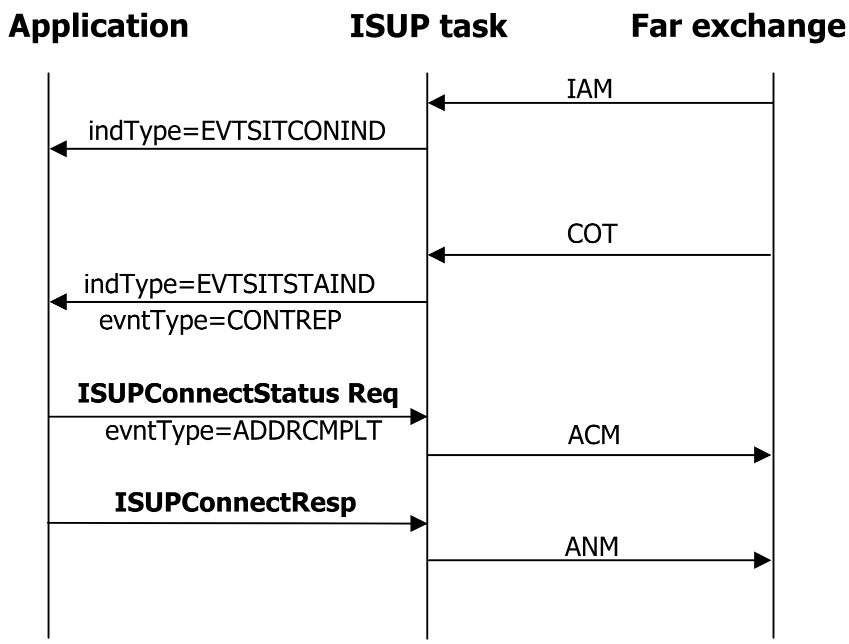


Alternatively, the far exchange can initiate the connection with continuity check required by sending the IAM message. The application receives a connect indication (EVTSITCONIND) event with the Nature of connection indicators IE field contChk set to CONTCHK_REQ. This is an indication to the application to set up a loopback condition on the specified circuit.

The far exchange then performs the continuity check and sends a continuity message (COT) to report the results of the check. This results in the generation of a status indication (EVTSITSTAIND) with event type CONTREP to the application.

Regardless of the results of the check, the application can remove the loop back condition on the circuit. If the continuity check was successful (status event structure contInd IE set to CONTCHK_SUCC), the application can continue with normal call setup. In the case of failure (status event structure contInd IE set to CONTCHK_FAIL), the application can wait for a continuity recheck on the indicated circuit.

The following illustration shows the functions and events used to establish an incoming connection with continuity check required:



Continuity recheck

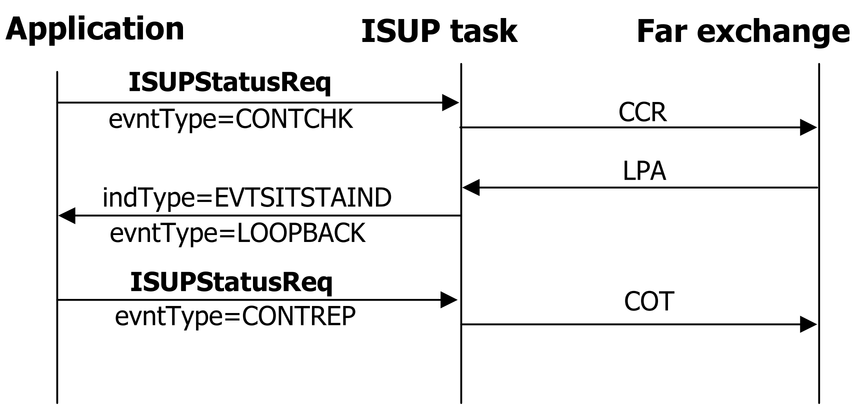
The application initiates a continuity recheck by invoking **ISUPStatusReq** with the event type of CONTCHK. This causes the ISUP layer to send a continuity check request message (CCR) to the far exchange.

The far exchange sets the indicated circuit in a loop back condition and acknowledges with a loop back acknowledgment message (LPA). Upon receipt of this message, the ISUP layer issues a status indication (EVTSITSTAIND) to the application with event type of LOOPBACK. The application can then perform the continuity check.

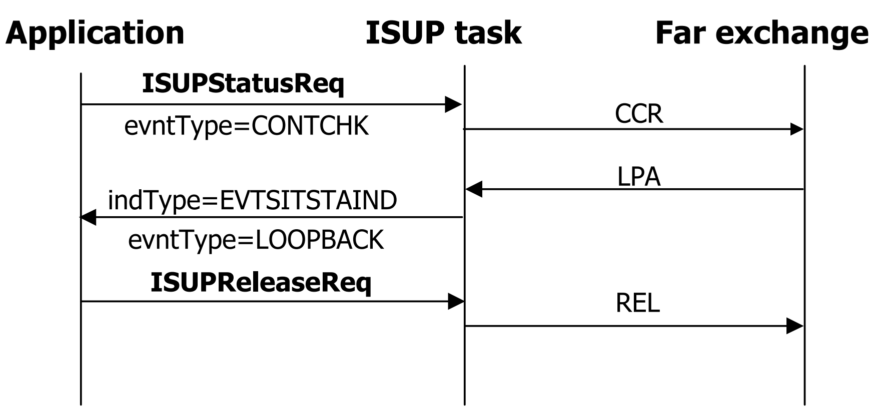
If the continuity check fails, the application can invoke **ISUPStatusReq** with the event type of CONTREP.

Note: The BICC variant does not support continuity checks or rechecks.

The following illustration shows an outgoing continuity recheck failure:



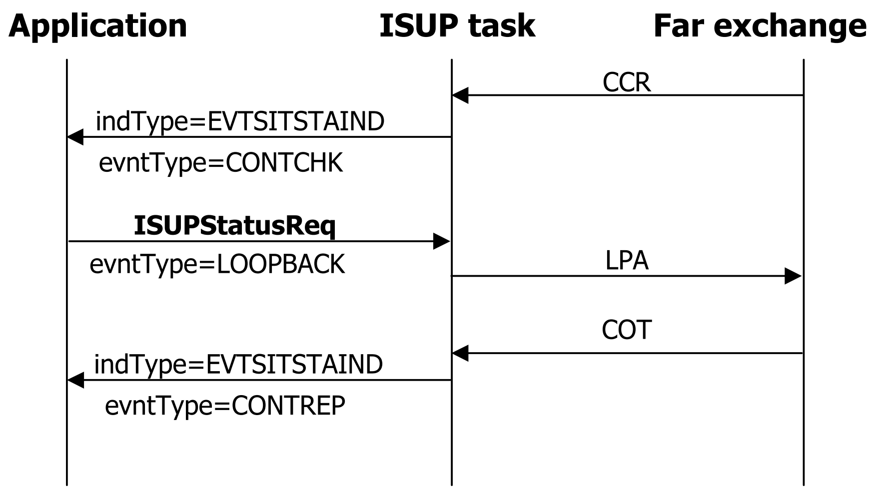
If the continuity check is successful, the application can release the connection by following the procedures for outgoing release as described in *Clearing connections* on page 30. The following illustration shows an outgoing continuity recheck success:



When a continuity check request message (CCR) is received from the far exchange, the ISUP layer issues a status indication (EVTSITSTAIND) with event type of CONTCHK to the application. The application can set the specified circuit into a loop back condition and acknowledge the request by invoking **ISUPStatusReq** with event type of LOOPBACK.

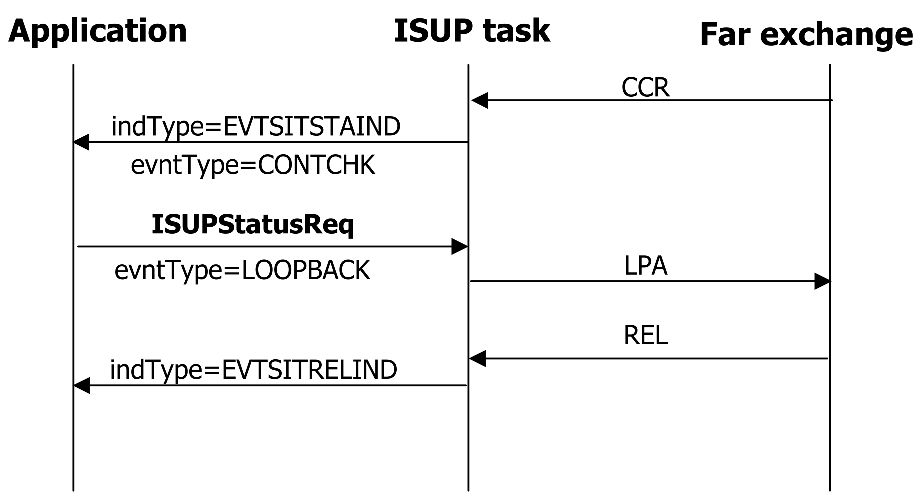
If the continuity test fails, the far exchange reports the results with a continuity message (COT). The ISUP layer reports the results to the application as a status indication (EVTSITSTAIND) with event type of CONTREP.

The following illustration shows an incoming continuity recheck failure:



If successful, the far exchange transmits a release message (REL) as a release indication (EVTSITRELIND) to the application. The application follows the normal procedures for a far exchange initiated release as described in *Clearing connections* on page 30.

The following illustration shows an incoming continuity recheck success:



Transferring data

While a call is connected, the application can exchange user-to-user information with its peer in the far exchange by invoking **ISUPDataReq** and/or receiving data indications (EVTSITDATIND) from the ISUP layer.

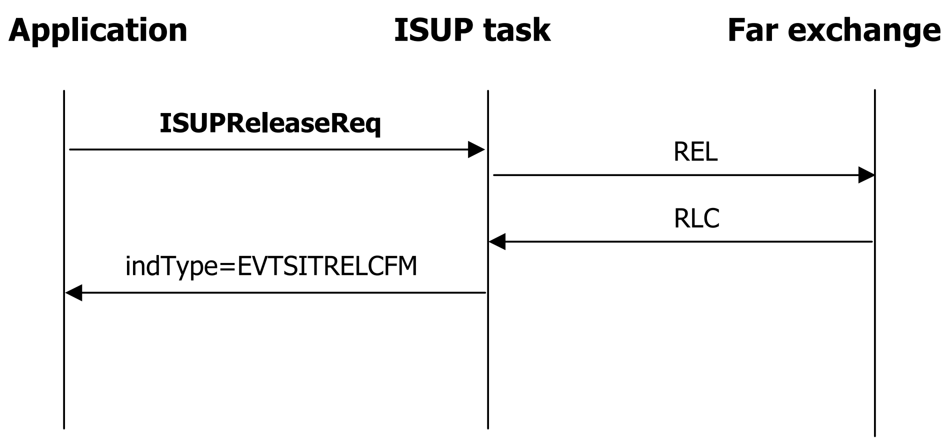
Either party can suspend the circuit connection by invoking **ISUPSuspendReq** or receiving a suspend indication (EVTSITSUSPIND) event.

Either party can resume the circuit connection by invoking **ISUPResumeReq** or receiving a resume indication (EVTSITRESMIND) event.

Clearing connections

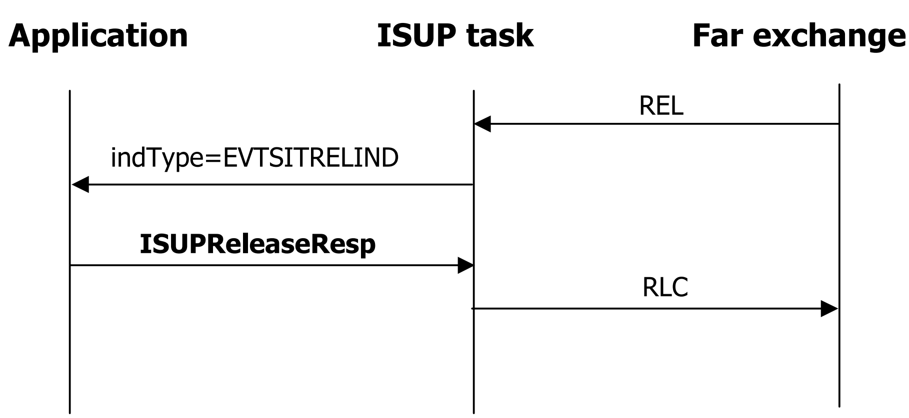
The application requests clearing of a connection by invoking **ISUPReleaseReq**. The application is notified of the completion of the release procedure (the receipt of a release complete message) by receipt of a release confirm (EVTRELCFM) event.

The following illustration shows a connection clearing request initiated by the application:



If the far exchange initiates the release of the connection, the application receives a release indication (EVTSITRELIND) event from the ISUP layer. The application then completes the connection release by invoking **ISUPReleaseResp** to send the release complete message to the far exchange.

The following illustration shows a connection clearing request initiated by the far exchange:

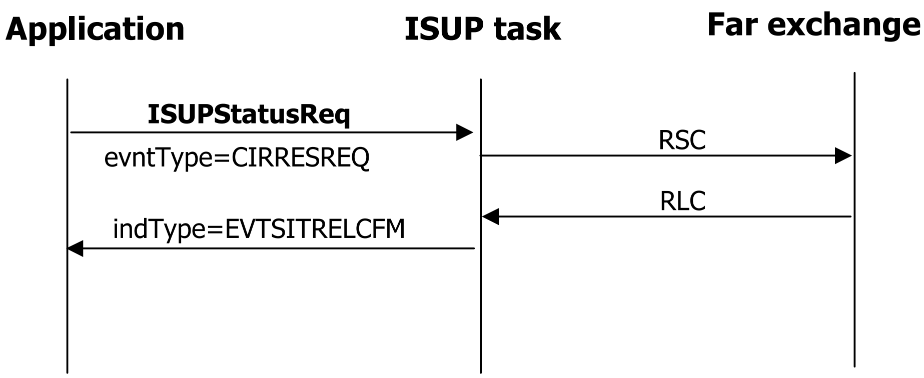


Resetting circuits

The application requests the reset of a circuit by invoking **ISUPStatusReq** with the event type of CIRRESREQ.

When an acknowledgment (release complete message) is received from the far exchange, the ISUP layer delivers a release confirm (EVTSITRELCFM) event to the application. The application can consider the circuit reset upon receiving this indication.

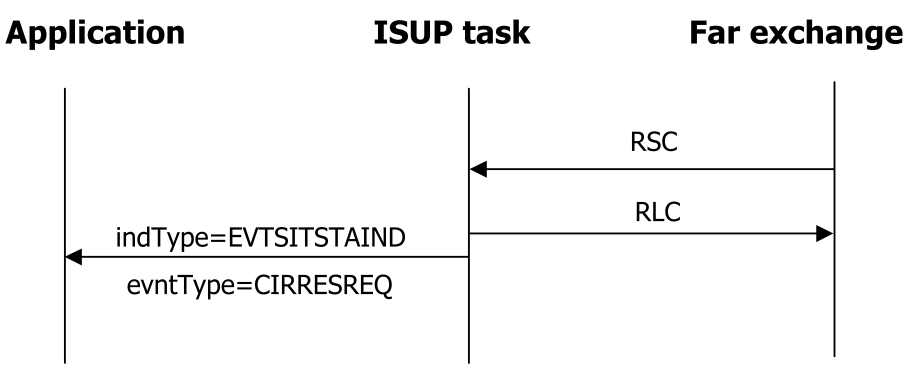
The following illustration shows a circuit reset initiated by the far exchange:



If the far exchange initiates the reset of the circuit, the application receives a status indication (EVTSITSTAIND) with the event type of CIRRESREQ from the ISUP layer. The application can consider the circuit reset upon receiving this indication. The ISUP task acknowledges the reset request by sending a release complete message (RLC).

Note: If the application issued any request related to this circuit that was not processed by the ISUP layer before receipt of the circuit reset message, the application receives an error indication (EVTSITERRIND) with the cause code CCINVALCALLREF. The application can choose to ignore this indication.

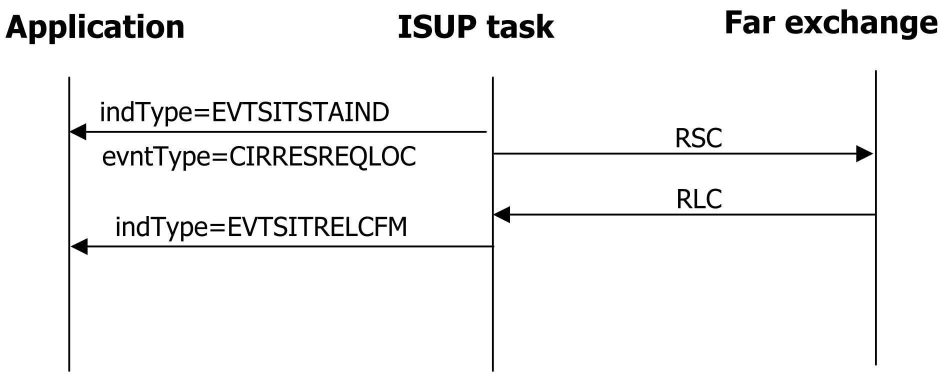
The following illustration shows a circuit reset initiated by the far exchange:



The ISUP layer can decide to reset a particular circuit due to protocol errors. Under these circumstances, the ISUP layer issues a status indication (EVTSITSTAIND) with the event type of CIRRESREQLOC, and sends a circuit reset message to the far exchange.

When an acknowledgment (release complete message) is received from the far exchange, the ISUP layer delivers a release confirm (EVTSITRELCFM) event to the application. The application can consider the circuit reset upon receiving this indication.

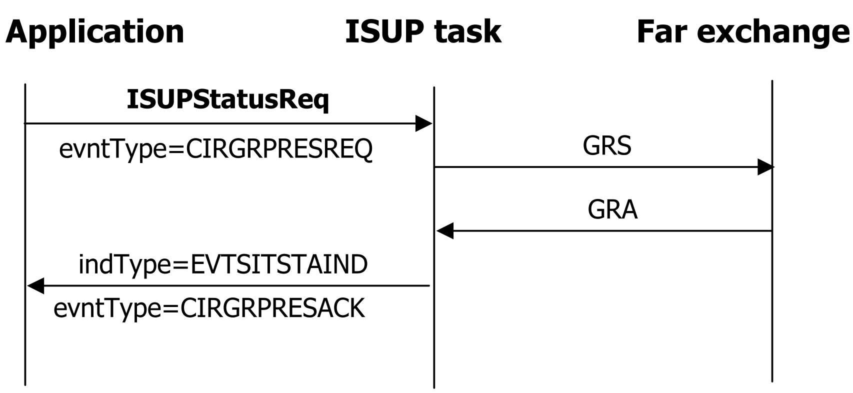
The following illustration shows a circuit reset initiated by the ISUP layer:



Resetting circuit groups

The application requests the reset of a circuit group by invoking **ISUPStatusReq** with the event type of CIRGRPRESREQ. The application is notified of the completion of the group reset procedure (the receipt of a group reset acknowledgment message) by receipt of a status indication (EVTSITSTAIND) with the event type of CIRGRPRESACK.

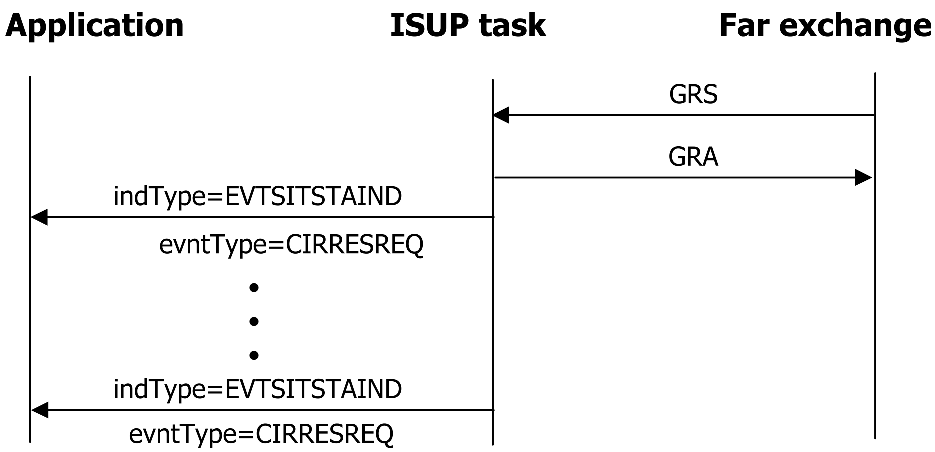
The following illustration shows a circuit group reset initiated by the application:



If the far exchange initiates the reset of the circuit group, the application receives a status indication (EVTSITSTAIND) with the event type of CIRRESREQ from the ISUP layer for each circuit in the circuit group. The application can consider these circuits reset upon receiving these indications. The ISUP task acknowledges the group reset request by sending a group reset acknowledgment message (GRA).

Note: To receive a single CIRGRPRESREQ, set the grpResetEvent parameter to TRUE in the general configuration parameters (**isupInitGenCfg**).

The following illustration shows a circuit group reset initiated by the far exchange:

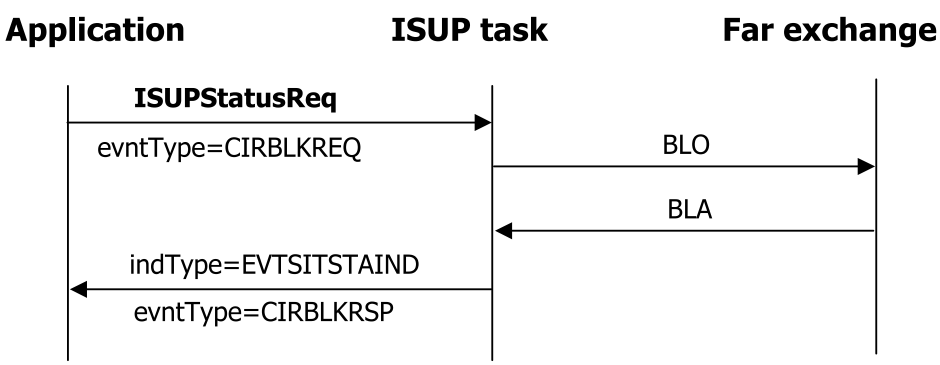


Blocking or unblocking circuits

The application requests blocking of a circuit by invoking **ISUPStatusReq** with the event type of CIRBLKREQ. The application is notified of the completion of the blocking procedure (the receipt of a blocking acknowledgment message) by receipt of a status indication (EVTSITSTAIND) with the event type of CIRBLKRSP.

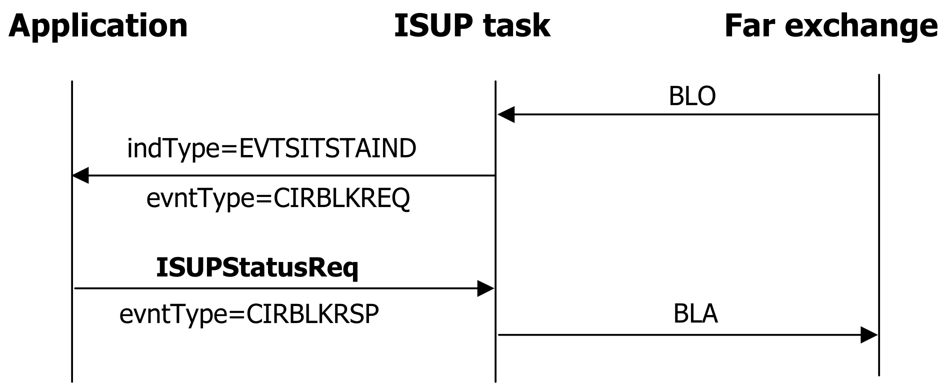
Note: The BICC variant does not support blocking or unblocking circuits.

The following illustration shows a blocking request initiated by the application:



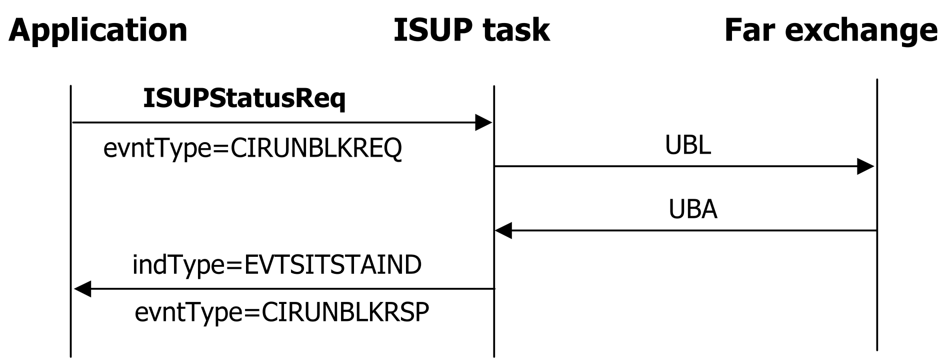
If the far exchange initiates the blocking of the circuit, the application receives a status indication (EVTSITSTAIND) with the event type of CIRBLKREQ from the ISUP layer. The application then acknowledges the circuit blocking by invoking **ISUPStatusReq** with the event type of CIRBLKRSP.

The following illustration shows a blocking request initiated by the far exchange:



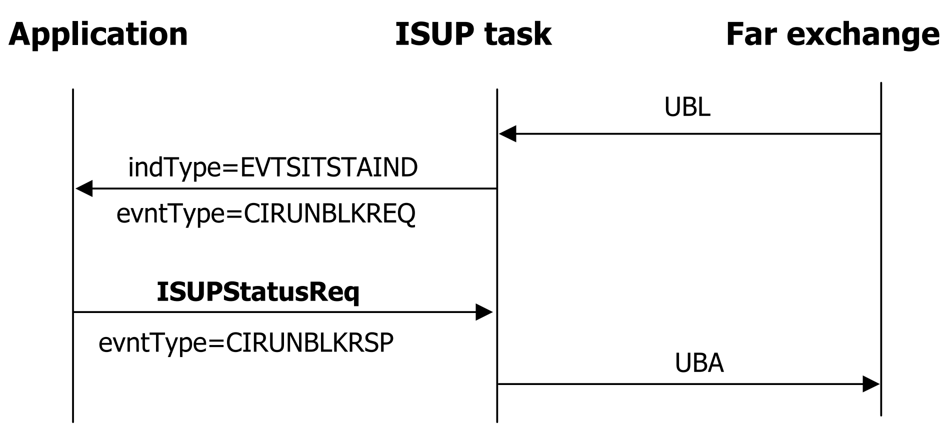
The application requests unblocking of a circuit by invoking **ISUPStatusReq** with the event type of CIRUNBLKREQ. The application is notified of the completion of the unblocking procedure (the receipt of an unblocking acknowledgment message) by receipt of a status indication (EVTSITSTAIND) with the event type of CIRUNBLKRSP.

The following illustration shows an unblocking request initiated by the application:



If the far exchange initiates the unblocking of the circuit, the application receives a status indication (EVTSITSTAIND) with the event type of CIRUNBLKREQ from the ISUP layer. The application then acknowledges the circuit unblocking by invoking **ISUPStatusReq** with the event type of CIRUNBLKRSP.

The following illustration shows an unblocking request initiated by the far exchange:



Blocking or unblocking circuit groups

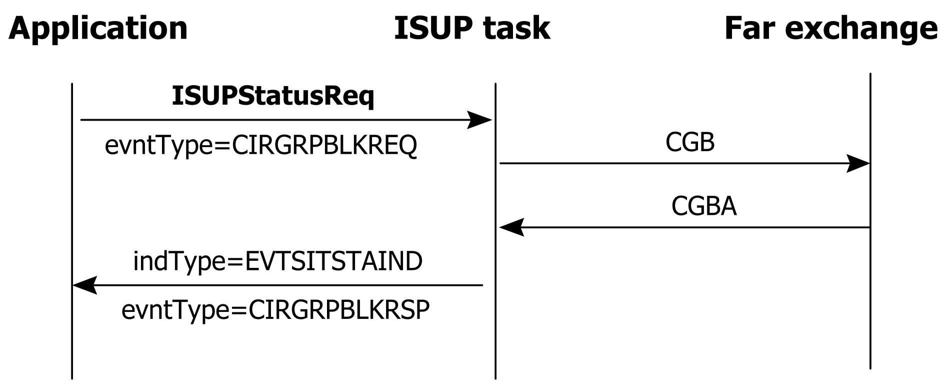
This topic provides information about clocking or unblocking circuit groups in the following scenarios:

- Group blocking request initiated by the application
- Group blocking request initiated by the far exchange
- Group unblocking request initiated by the application
- Group unblocking request initiated by the far exchange

Group blocking request initiated by the application

The application requests blocking of a circuit group by invoking **ISUPStatusReq** with the event type of CIRGRPBLKREQ. The application is notified of the completion of the group blocking procedure (the receipt of a group blocking acknowledgment message) by receipt of a status indication (EVTSITSTAIND) with the event type of CIRGRPBLKRSP.

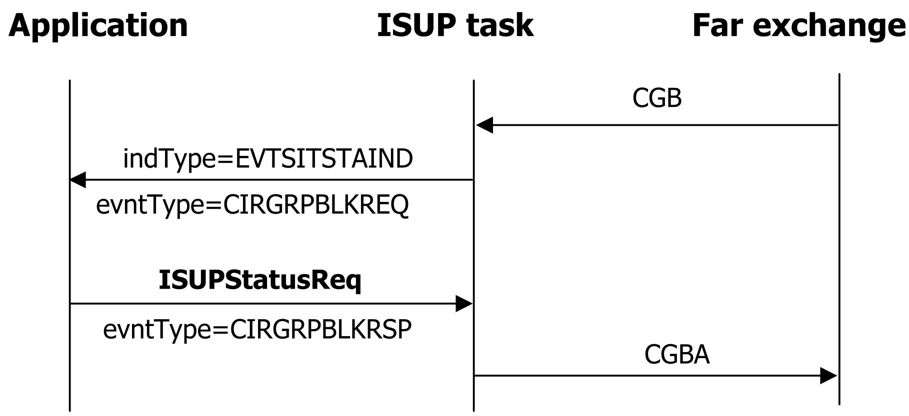
The following illustration shows a group blocking request initiated by the application:



Group blocking request initiated by the far exchange

If the far exchange initiates the blocking of the circuit group, the application receives a status indication (EVTSITSTAIND) with the event type of CIRGRPBLKREQ from the ISUP layer. The application acknowledges the circuit group blocking by invoking **ISUPStatusReq** with the event type of CIRGRPBLKRSP.

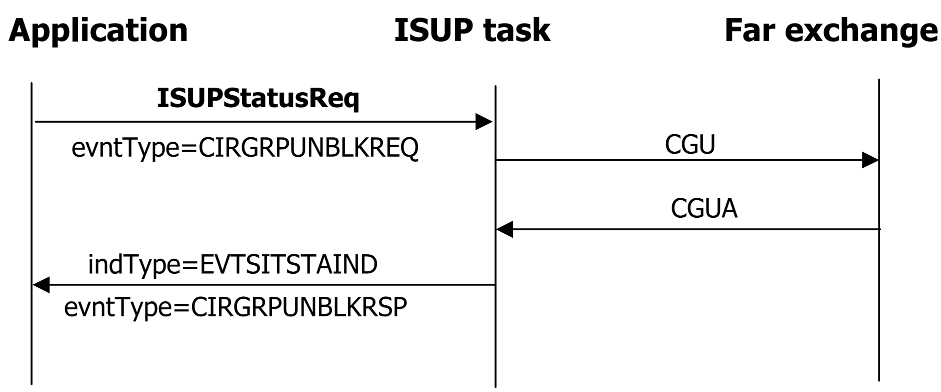
The following illustration shows a group blocking request initiated by the far exchange:



Group unblocking request initiated by the application

The application requests unblocking of a circuit group by invoking **ISUPStatusReq** with the event type of CIRGRPUNBLKREQ. The application is notified of the completion of the group unblocking procedure (the receipt of a group unblocking acknowledgment message) by receipt of a status indication (EVTSITSTAIND) with the event type of CIRGRPUNBLKRSP.

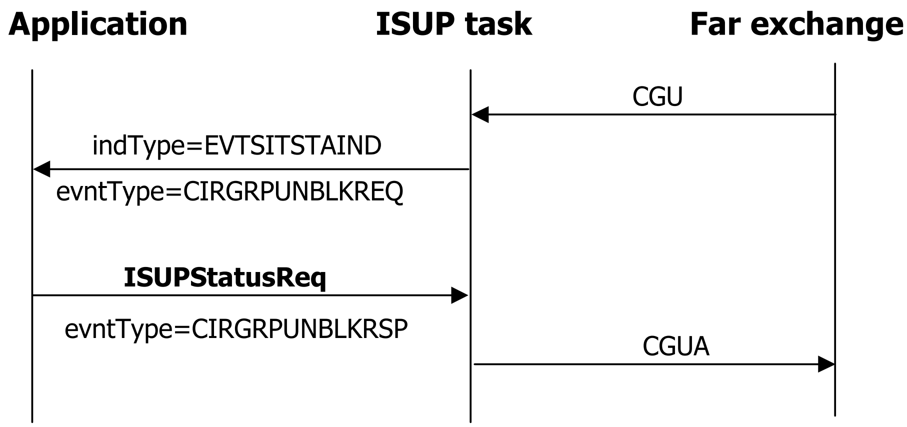
The following illustration shows a group unblocking request initiated by the application:



Group unblocking request initiated by the far exchange

If the far exchange initiates the unblocking of the circuit group, the application receives a status indication (EVTSITSTAIND) with the event type of CIRGRPUNBLKREQ from the ISUP layer. The application then acknowledges the circuit group unblocking by invoking **ISUPStatusReq** with the event type of CIRGRPUNBLKRSP.

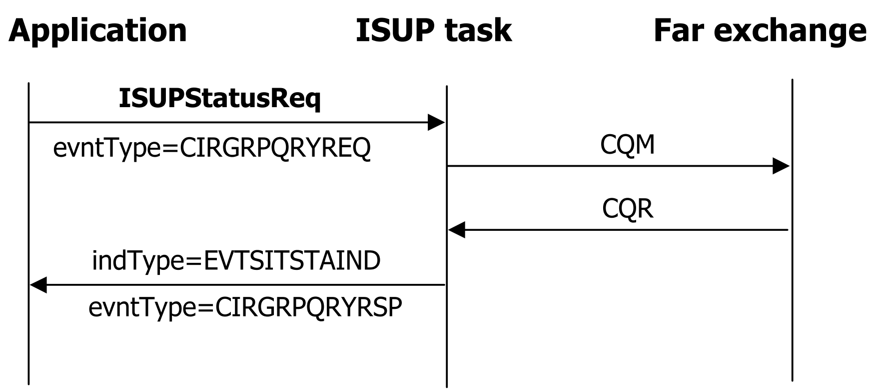
The following illustration shows a group unblocking request initiated by the far exchange:



Querying circuit groups

The application initiates a circuit group query by invoking **ISUPStatusReq** with the event type of CIRGRPQRYREQ. The application is notified of the completion of the group query request (the receipt of a circuit query response message) by receipt of a status indication (EVTSITSTAIND) with the event type of CIRGRPQRYRSP.

The following illustration shows a circuit group query request initiated by the application:



Checkpointing circuit states

ISUPStatusReq supports two event types for circuit state checkpointing. These event types are:

Event type	Description
CIRGRPSET	Circuit group set status.
CIRGRPGET	Circuit group get status

Both status request types use the range and status information element. For the purposes of the two new status request types, the range and status information element fields are used as follows:

Field	Description
range	Number of circuits to modify (minus one). For example, to modify 24 circuits, the range value is 23.
status	An array of length range + 1. Each byte of the array contains the circuit status for one circuit.

The circuit status is encoded and decoded as:

```
MS four bits    = circuit Blocking status
    BSNOTBLK    = Idle
    BSRMTBLK    = Remotely Blocked
    BSLOCBLK    = Locally Blocked
    BSLOCRMTBLK = Remotely and Locally Blocked

LS four bits = circuit Call status
    CSIDLE    = Idle
    CSINCBUSY = Incoming Busy
    CSOUTBUSY = Outgoing Busy
```

For example, if a circuit is locally blocked and the incoming circuit is busy, its status byte is encoded as 0x21. If it is not blocked and the outgoing circuit is busy, then its status byte is encoded as 0x02.

Controlling ISUP congestion

The ISUPEVN_CONG event indicates one of the following congestion issues:

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

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

If your application receives a level one event, reduce the number of calls being generated. At levels two and three, 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 normal traffic.

If your application requires additional information about the congestion, call **ISUPGetApiStats**.

Tracing function calls and events

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

```
[ctasys]
Service = isup, isupmgr
```

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

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

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

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

Handling redundancy events

After binding to an ISUP 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
MTPSTANDALONE	Application is in a non-redundant configuration. Normal operation can begin.
MTPPRIMARY	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.
MTPBACKUP	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.

5 ISUP service function reference

ISUP service function summary

The ISUP service provides the following asynchronous functions:

Connection establishment functions

Function	Description
ISUPConnectReq	Requests the establishment of a circuit switched connection.
ISUPConnectResp	Signals the far exchange that an incoming call was answered.
ISUPConnectStatusReq	Sends connection status information to the far exchange during the connection establishment phase.
ISUPStatusReq	Sends a global or circuit-specific message to the far exchange.

Data transfer functions

Function	Description
ISUPDataReq	Sends user-to-user information associated with an established connection to the far exchange.
ISUPResumeReq	Resumes a suspended connection (cancels timer T2) and sends a resume message to the far exchange.
ISUPSuspendReq	Sends a suspend message to the far exchange and starts the suspend timer T2.

Connection clearing functions

Function	Description
ISUPReleaseReq	Clears or denies the establishment of a circuit switched connection.
ISUPReleaseResp	Responds to a release indication from a far exchange.

Miscellaneous functions

Function	Description
ISUPFacilityReq	Sends a facility request message to the far exchange.
ISUPGetApiStats	Retrieves congestion level activity statistics from the ISUP service.
ISUPRawReq	Sends an application-encoded ISUP packet.
ISUPRetrieveMessage	Retrieves the next message from the ISUP layer.

Using the ISUP service function reference

This section provides an alphabetical reference to the ISUP service functions. A typical function includes:

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

ISUPConnectReq

Requests the establishment of a circuit switched connection.

Prototype

DWORD **ISUPConnectReq** (CTAHD ***ctahd***, SuId ***suId***, SiInstId ***suInstId***, SiInstId ***spInstId***, Bool ***cirSelFlg***, CirId ***circuit***, SiConEvnt ****conEvnt***)

Argument	Description
ctahd	Natural Access handle returned by **ctaCreateContext**.
suId	ISUP service access point.
suInstId	Service user instance ID.
spInstId	Service provider instance ID.
cirSelFlg	Circuit selection flag.
circuit	Circuit ID to be 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
ISUP_SUCCESS	
CTAERR_BAD_ARGUMENT	One or more arguments are invalid.
CTAERR_DRIVER_SEND_FAILED	Error occurred accessing the TX driver.
CTAERR_INVALID_CTAHD	Natural Access handle is invalid.

Details

When successful, **ISUPConnectReq** results in an initial address message (IAM) being sent to the far exchange. If the ISUP layer cannot successfully initiate the outgoing connection request (for example, due to network congestion or because the requested is not idle), it returns an asynchronous status indication event to the application with the cause value coded with the reason for the failure.

The value specified in ***suInstId*** is passed on to all subsequent events associated with this connection.

If this message is associated with a previously established connection, such as in a circuit reservation or continuity test, the ***spInstId*** must be the ***spInstId*** value returned in the first event received from the ISUP layer relative to this connection. If this is the first message associated with this connection, this value is coded to zero.

Set the value for ***cirSelFlg*** to non-zero (true) since the application must select the circuit.

Example

```
#define SAP_ID 0

U8          cdPty[20] = "8479258900";
U8          cdPtyLen = 0;
U8          *calling = NULL;      /* default = no calling pty info in IAM */
U8          cgPty[20];
U8          cgPtyLen = 0;
SiConEvnt   siConEvnt;
CTAHD       FstCtaHd = Valid CTA Handle;

S16         switchType = ST_ITUWHITE;
SiInstId    suInstId = 0;
SiInstId    spInstId = 0;
SuId        suId = SAP_ID;
CirId       circuit = 2;  /* Placing call on circuit 2 */

cdPtyLen = ISUPASCIItoBCD(called, cdPty, 20);
cgPtyLen = ISUPASCIItoBCD(calling, cgPty, 20);

printf("cdPtyLen = %d, cgPtyLen = %d\n", cdPtyLen, cgPtyLen);

ISUPInitIAM(switchType, &siConEvnt, cdPty, cdPtyLen, cgPty, cgPtyLen);
     status = ISUPConnectReq(FstCtaHd, suId, suInstId, spInstId, 1, circuit, &siConEvnt);
     if( status != ISUP_SUCCESS )
         printf( "ISUPConnectReq() failed status = %d\n", status );
     else
          printf( "Initial Address Message sent for circuit %ld\n", circuit )
```

ISUPConnectResp

Signals the far exchange that an incoming call was answered.

Prototype

DWORD **ISUPConnectResp** (CTAHD ***ctahd***, SuId ***suId***, SiInstId ***suInstId***, SiInstId ***spInstId***, CirId ***circuit***, SiConEvnt ****conEvnt***)

Argument	Description
ctahd	Natural Access handle returned by **ctaCreateContext**.
suId	ISUP service access point.
suInstId	Service user instance ID.
spInstId	Service provider instance ID.
circuit	Circuit with which this message is associated.
conEvnt	Pointer to the caller's connect event structure containing all parameters (IEs) included in the ANSWER (CONNECT) message.

Return values

Return value	Description
ISUP_SUCCESS	
CTAERR_BAD_ARGUMENT	One or more arguments are invalid.
CTAERR_DRIVER_SEND_FAILED	Error occurred accessing the TX driver.
CTAERR_INVALID_CTAHD	Natural Access handle is invalid.

Details

ISUPConnectResp generates an answer (ANM) or connect (ITU-T only) message to the far exchange.

The value specified in ***suInstId*** is passed to all subsequent events associated with this connection.

The value for ***spInstId*** must be the ***spInstId*** value that was received from the ISUP layer in the connect indication event.

Example

In this example, an incoming call is answered by sending an ANM with **ISUPConnectResp**.

```
DWORD           status;
SiAllSdus       rcvEvent;
IsupRcvInfoBlk  rcvInfo;
S16             switchType = ST_ITUWHITE;

status = ISUPRetrieveMessage(ctahd, &rcvEvent, &rcvInfo, 1);
if (status == ISUP_NOMSG)
{
    fprintf(stderr, "ISUPRetrieveMessage() did not get a message, probably a
        congestion event \n");
    return(ISUP_SUCCESS);
}
    if (status != ISUP_SUCCESS)
    {
        fprintf(stderr, "ISUPRetrieveMessage() failed, returned %d\n", status);
        return(status);
    }
...
...
/*Sending ACM in response to incoming call */
...
...
/* The incoming call is answered by calling ISUPConnectResp */
/* send answer message (ANM) */
ISUPInitANM(switchType, &rcvEvent.m.siConEvnt);
status = ISUPConnectResp(ctahd, rcvInfo.suId, rcvInfo.suInstId, rcvInfo.spInstId,
         rcvInfo.circuit, &rcvEvent.m.siConEvnt);
if (status != ISUP_SUCCESS)
    printf("term: ISUPConnectResp() failed status = %d\n", status)
else
    printf("term: Answer sent for circuit %ld\n", rcvInfo.circuit);
```

ISUPConnectStatusReq

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

Prototype

DWORD **ISUPConnectStatusReq** (CTAHD ***ctahd***, SuId ***suId***, SiInstId ***suInstId***, SiInstId ***spInstId***, CirId ***circuit***, SiConStEvnt ****conStEvnt***, U8 ***eventType***)

Argument	Description
ctahd	Natural Access handle returned by **ctaCreateContext**.
suId	ISUP service access point.
suInstId	Service user instance ID.
spInstId	Service provider instance ID.
circuit	Circuit 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.
eventType	Identifies the type of message sent to the far exchange: ADDRCMPLT = Address complete MODIFY = Call modification request MODCMPLT = Call modification complete MODREJ = Call modification rejected PROGRESS = Call progress information FRWDTRSFR = Forward transfer IDENTREQ = Identification request IDENTRSP = Identification response INFORMATION = Information (response to INFORMATREQ) INFORMATREQ = Information request SUBSADDR = Subsequent address message NETRESMGR = Network resource manager

Return values

Return value	Description
ISUP_SUCCESS	
CTAERR_BAD_ARGUMENT	One or more arguments are invalid.
CTAERR_DRIVER_SEND_FAILED	Error occurred accessing the TX driver.
CTAERR_INVALID_CTAHD	Natural Access handle is invalid.

Details

The connection status information can be address complete, progress, information request. For more information, refer to *Establishing connections* on page 24.

The following NTT-specific value can be passed as the evntType argument to **ISUPConnectStatusReq**:

CHARGE ***Charge Message***

The value specified in the ***suInstId*** field is passed to all subsequent events associated with this connection.

The value for ***spInstId*** must be the ***spInstId*** value that was received from the ISUP layer in the connect indication event.

Example

In this example, the incoming call is accepted by sending an ACM with **ISUPConnectStatusReq**:

```
DWORD status;
SiAllSdus       rcvEvent;
IsupRcvInfoBlk  rcvInfo;
S16 switchType = ST_ITUWHITE;

status = ISUPRetrieveMessage(ctahd, &rcvEvent, &rcvInfo, 1);
if (status == ISUP_NOMSG)
    {

 fprintf(stderr, "ISUPRetrieveMessage() did not get a message, probably a congestion
         event \n");
         return(ISUP_SUCCESS);
    }
if (status != ISUP_SUCCESS)
    {
         fprintf(stderr, "ISUPRetrieveMessage() failed, returned %d\n", status);
         return(status);
    }
/* determine indication/confirmation type received */
switch(rcvInfo.indType)
{
    ...
    ...
    case EVTSITCONIND: /* connect indication (incoming call) */
        printf("term: Connect Indication received for circuit %ld, ", rcvInfo.circuit);
        /* send address complete message (ACM) */
        ISUPInitACM(switchType, &rcvEvent.m.siCnStEvnt);
        status = ISUPConnectStatusReq(ctahd, rcvInfo.suId, rcvInfo.suInstId,
            rcvInfo.spInstId, rcvInfo.circuit, &rcvEvent.m.siCnStEvnt, ADDRCMPLT);
         if (status != ISUP_SUCCESS)
             printf("term: ISUPReleaseResp() failed status = %d\n", status);
         else
             printf("term: Address Complete sent for circuit %ld\n", rcvInfo.circuit);
}
```

ISUPDataReq

Sends user-to-user information associated with an established connection to the far exchange.

Prototype

DWORD **ISUPDataReq** (CTAHD ***ctahd***, SuId ***suId***, SiInstId ***suInstId***, SiInstId ***spInstId***, CirId ***circuit***, SiInfoEvnt ****infoEvnt***)

Argument	Description
ctahd	Natural Access handle returned by **ctaCreateContext**.
suId	ISUP service access point.
suInstId	Service user instance ID.
spInstId	Service provider instance ID.
circuit	The circuit with which this message is associated.
infoEvnt	Pointer to the caller's information event structure containing all parameters (IEs) and user-to-user data included in the message to the far exchange.

Return values

Return value	Description
ISUP_SUCCESS	
CTAERR_BAD_ARGUMENT	One or more arguments are invalid.
CTAERR_DRIVER_SEND_FAILED	Error occurred accessing the TX driver.
CTAERR_INVALID_CTAHD	Natural Access handle is invalid.

Details

The value specified in the ***suInstId*** field is passed to all subsequent events associated with this connection.

The value for ***spInstId*** must be the ***spInstId*** value that was received from the ISUP layer in the connect indication event.

Example

```
#define SAP_ID 0

CTAHD      FstCtaHd = Valid CTA Handle;
DWORD      status;
SiInstId   suInstId = 0;
SiInstId   spInstId = 0;
SuId       suId = SAP_ID;
CirId      circuit = 2;
SiAllSdus  sendBuffer;

status = = ISUPDataReq( FstCtaHd, suId, suInstId, spInstId, circuit,
    sendBuffer.m.siInfoEvnt );
if( status != ISUP_SUCCESS )
    {
        printf( "ERROR: ISUPDataReq( circuit %d ) failed [%d]", circuit, status );
        return( -1 );
    }
```

ISUPFacilityReq

Sends a facility request message to the far exchange.

Prototype

DWORD **ISUPFacilityReq** (CTAHD ***ctahd***, SuId ***suId***, SiInstId ***suInstId***, SiInstId ***spInstId***, CirId ***circuit***, SiFacEvnt ****facEvnt***, U8 ***eventType***)

Argument	Description
ctahd	Natural Access handle returned by **ctaCreateContext**.
suId	ISUP service access point.
suInstId	Service user instance ID.
spInstId	Service provider instance ID.
circuit	Circuit with which this message is associated.
facEvnt	Pointer to the caller's facility event structure containing all parameters (IEs) included in the message to the far exchange.
eventType	Type of facility request. Refer to the Details section for a list of values.

Return values

Return value	Description
ISUP_SUCCESS	
CTAERR_BAD_ARGUMENT	One or more arguments are invalid.
CTAERR_DRIVER_SEND_FAILED	Error occurred accessing the TX driver.
CTAERR_INVALID_CTAHD	Natural Access handle is invalid.

Details

The value specified in the ***suInstId*** field is passed to all subsequent events associated with this connection.

The value for ***spInstId*** must be the ***spInstId*** value that was received from the ISUP layer in the connect indication event.

Possible values for ***eventType*** are:

Value	Description
FACILITY	Facility
FACILITYREQ	Facility request
FACILITYACC	Facility accept
FACILITYREJ	Facility reject
FACILITYDEACT	Facility deactivate
FACILITYINFO	Facility information

Example

```
#define SAP_ID 0

CTAHD      FstCtaHd = Valid CTA Handle;
DWORD      status;
SiInstId   suInstId = 0;
SiInstId   spInstId = 0;
SuId       suId = SAP_ID;
CirId      circuit = 2;
U8         evntType =  FACILITYREQ;
SiAllSdus  sendBuffer;

status = ISUPFacilityReq( FstCtaHd, suId,  suInstId,  spInstId,  circuit,
    sendBuffer.m.siFacEvnt, evntType );
if( status != ISUP_SUCCESS )
    {
        printf( "ERROR: ISUPFacilityReq( circuit %d, type %d ) failed [%d]", circuit,
            evntType, status );
        return( -1 );
    }
```

ISUPGetApiStats

Retrieves congestion level activity statistics from the ISUP service.

Prototype

DWORD **ISUPGetApiStats** (CTAHD ***ctahd***, ISUPAPISTATS ****pStats***, U8 ***reset***)

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

pStats	Pointer to the address of the buffer where statistics are returned to the caller:
	`typedef struct`
	`{`
	`U32 qCount; /* Number of API messages currently * queued to ISUP layer */`
	`U32 qPeak; /* Max number of API messages ever * queued to ISUP layer */`
	`U32 txPending; /* Current number of outstanding transmit * rqsts to ISUP layer */`
	`U32 txPendPeak; /* Max number of transmit rqsts ever * outstanding to ISUP 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 ISUP * layer */`
	`U8 apiQCongLvl; /* Current outbound queue congestion * level [0..3] */`
	`U8 isupCongLvl; /* Current ISUP layer congestion * level [0..3] */`
	`U8 isupCongSrc; /* Reason for ISUP layer congestion */`
	`U8 spare1; /* Spare for alignment */`
	`} ISUPAPISTATS;`

reset	If non-zero, statistics are reset after returning the statistics to the application.

Return values

Return value	Description
ISUP_SUCCESS	
CTAERR_INVALID_CTAHD	Natural Access handle is invalid.

Example

```
ISUPAPISTATS  pStats;
DWORD         status;
CTAHD         FstCtaHd = Valid CTA Handle;


status = ISUPGetApiStats(FstCtaHd, &pStats, 0);
if (status != ISUP_SUCCESS)
    printf("isuporig: ISUPGetApiStats() failed status = %d\n", status);
    else
    {
        printf("qCount = %x\n", pStats.qCount);
        printf("qPeak = %x\n", pStats.qPeak);
        printf("txPending = %x\n", pStats.txPending);
        printf("txPendPeak = %x\n", pStats.txPendPeak);
        printf("txSuccess = %x\n", pStats.txSuccess);
        printf("txFailed = %x\n", pStats.txFailed);
        printf("txLastErr = %x\n", pStats.txLastErr);
        printf("rxSuccess = %x\n", pStats.rxSuccess);
        printf("apiQCongLvl = %x\n", pStats.apiQCongLvl);
        printf("isupCongLvl = %x\n", pStats.isupCongLvl);
        printf("isupCongSrc = %x\n", pStats.isupCongSrc);
     }
```

ISUPRawReq

Sends an application-encoded ISUP packet.

Prototype

DWORD **ISUPRawReq** (CTAHD ***ctahd***, SuId ***suId***, SiInstId ***suInstId***, SiInstId ***spInstId***, Bool ***cirSelFlg***, CirId ***circuit***, SiRawEvnt ****rawEvnt***, U8 ***newState***)

Argument	Description
ctahd	Natural Access handle returned by **ctaCreateContext**.
suId	ISUP service access point.
suInstId	Service user instance ID.
spInstId	Service provider instance ID.
cirSelFlg	Circuit selector flag. This value must always be FALSE.
circuit	Circuit index with which this message is associated.
rawEvnt	Pointer to caller's raw event structure.
newState	New state of circuit.

Return values

Return value	Description
ISUP_SUCCESS	
CTAERR_BAD_ARGUMENT	One or more arguments are invalid.
CTAERR_DRIVER_SEND_FAILED	Error occurred accessing the TX driver.
CTAERR_INVALID_CTAHD	Natural Access handle is invalid.

Details

An application can receive unknown message types as raw messages and can send raw messages.

ISUPRawReq enables an application to build its own ISUP packets. The application must encode the entire message starting from the message type field. The ISUP task builds the routing label and places the CIC code on the front of the message. The TX software changes the state of the circuit to reflect that of the ***newState*** parameter that has been sent by the application and sends out the message as defined by the application in the raw event structure. Valid ***newState*** values are presented in the following table:

Value	Description
RAWST_NOCHANGE (1)	Leaves the circuit in its current state.
RAWST_IDLE (2)	Changes the circuit to not busy.
RAWST_BUSYIN (3)	Changes the circuit state to answered for an inbound call.
RAWST_BUSYOUT (4)	Changes the circuit state to answered for an outbound call.
RAWST_WAITACM (5)	Changes the circuit state to waiting for an ACM, typically after a (non-standard) IAM is sent.

The TX board passes the raw data as a new event type to the application in the new raw event structure, when it receives a message with an unrecognized message type. The first byte the board places in the data area of the new raw message event structure is the message type. For example, the first byte is 0x01 for a traditional IAM packet. This value is not checked by the software and can be any value. The ISUP service does not change the circuit's state based on any of these unknown messages.

Example

```
#define SAP_ID 0

CTAHD      FstCtaHd = Valid CTA Handle;
DWORD      status;
SiInstId   suInstId = 0;
SiInstId   spInstId = 0;
SuId       suId = SAP_ID;
CirId      circuit = 2;
SiAllSdus  sendBuffer;

status = ISUPRawReq( FstCtaHd, suId,  suInstId,  spInstId,  circuit,
    &sendBuffer.m.siRawEvnt);
if( status != ISUP_SUCCESS )
    {
         printf( "ERROR: ISUPRawReq( circuit %d ) failed [%d]", circuit, status );
             return( -1 );
    }
```

ISUPReleaseReq

Clears or denies the establishment of a circuit switched connection.

Prototype

DWORD **ISUPReleaseReq** (CTAHD ***ctahd***, SuId ***suId***, SiInstId ***suInstId***, SiInstId ***spInstId***, CirId ***circuit***, SiRelEvnt ****relEvnt***)

Argument	Description
ctahd	Natural Access handle returned by **ctaCreateContext**.
suId	ISUP service access point.
suInstId	Service user instance ID.
spInstId	Service provider instance ID.
circuit	Circuit with which this message is associated.
relEvnt	Pointer to the caller's release event structure containing all parameters (IEs) included in the message to the far exchange.

Return values

Return value	Description
ISUP_SUCCESS	
CTAERR_BAD_ARGUMENT	One or more arguments are invalid.
CTAERR_DRIVER_SEND_FAILED	Error occurred accessing the TX driver.
CTAERR_INVALID_CTAHD	Natural Access handle is invalid.

Details

ISUPReleaseReq generates a release message to the far exchange.

The value specified in the ***suInstId*** field is passed to all subsequent events associated with this connection.

The value for ***spInstId*** must be the ***spInstId*** value that was received from the ISUP layer in the connect indication event.

Example

```
DWORD status;
SiAllSdus  rcvEvent;
SiAllSdus  sendEvent;
IsupRcvInfoBlk  rcvInfo;
CTAHD  ctahd = valid CTA handle;

/* Handling all incoming ISUP messages */
status = ISUPRetrieveMessage(ctahd, &rcvEvent, &rcvInfo, 1);
if (status == ISUP_NOMSG)
    {
        fprintf(stderr, "ISUPRetrieveMessage() did not get a message, probably
            a congestion event \n");
            return(ISUP_SUCCESS);
    }
if (status != ISUP_SUCCESS)
    {
        fprintf(stderr, "ISUPRetrieveMessage() failed, returned %d\n", status);
        return(status);
    }

/* determine indication/confirmation type received */
switch( rcvInfo.indType )
{
      ...
      ...
    case EVTSITCONCFM:              /* connect confirmation */
    printf("orig: Connect Confirmation received for circuit %ld\n",  rcvInfo.circuit );

    /* Releasing the call */
    ISUPInitREL(switchType, &sendEvent.m.siRelEvnt, CCCALLCLR);
    status = ISUPReleaseReq(ctahd, rcvInfo.suId, rcvInfo.suInstId,  rcvInfo.spInstId,
        rcvInfo.circuit, &sendEvent.m.siRelEvnt);
    if (status != ISUP_SUCCESS)
        printf("orig: ISUPReleaseReq() failed status = %d\n", status);
    else
        printf("orig: Release sent for circuit %ld\n", rcvInfo.circuit);
        break;

        case EVTSITRELCFM:              /* Release confirmation (release complt) */
            ...
            ...

}
```

ISUPReleaseResp

Responds to a release indication from a far exchange.

Prototype

DWORD **ISUPReleaseResp** (CTAHD ***ctahd***, SuId ***suId***, SiInstId ***suInstId***, SiInstId ***spInstId***, CirId ***circuit***, SiRelEvnt ****relEvnt***)

Argument	Description
ctahd	Natural Access handle returned by **ctaCreateContext**.
suId	ISUP service access point.
suInstId	Service user instance ID.
spInstId	Service provider instance ID.
circuit	Circuit 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
ISUP_SUCCESS	
CTAERR_BAD_ARGUMENT	One or more arguments are invalid.
CTAERR_DRIVER_SEND_FAILED	Error occurred accessing the TX driver.
CTAERR_INVALID_CTAHD	Natural Access handle is invalid.

Details

ISUPReleaseResp sends a release complete message to the far exchange and makes the circuit available for a new connection in the ISUP circuit database.

The value specified in the ***suInstId*** field is passed to all subsequent events associated with this connection.

The value for ***spInstId*** must be the ***spInstId*** value that was received from the ISUP layer in the connect indication event.

Example

```
DWORD status;
SiAllSdus       rcvEvent;
SiAllSdus       sendEvent;
IsupRcvInfoBlk  rcvInfo;

/* Handling all incoming ISUP messages */
status = ISUPRetrieveMessage(ctahd, &rcvEvent, &rcvInfo, 1);
if (status == ISUP_NOMSG)
    {
         fprintf(stderr, "ISUPRetrieveMessage() did not get a message, probably a
             congestion event \n");
         return(ISUP_SUCCESS);
   }
if (status != ISUP_SUCCESS)
    {
         fprintf(stderr, "ISUPRetrieveMessage() failed, returned %d\n", status);
         return(status);
    }

/* determine indication/confirmation type received */
switch( rcvInfo.indType )
{
    ...
    ...
    case EVTSITRELIND:              /* Release indication */
        printf("Release Indication for circuit %ld\n",   rcvInfo.circuit);

        status = ISUPReleaseResp(ctahd, rcvInfo.suId, rcvInfo.suInstId,
            rcvInfo.spInstId, rcvInfo.circuit, &rcvEvent.m.siRelEvnt);
        if (status != ISUP_SUCCESS)
            printf("orig: ISUPReleaseResp() failed status = %d\n", status);
        else
            printf("orig: Release Complete sent for circuit %ld\n", rcvInfo.circuit);
            break;

    case EVTSITRELCFM:              /* Release confirmation (release complete) */
        ...
        ...

}
```

ISUPResumeReq

Resumes a suspended connection (cancels timer T2) and sends a resume message to the far exchange.

Prototype

DWORD **ISUPResumeReq** (CTAHD ***ctahd***, SuId ***suId***, SiInstId ***suInstId***, SiInstId ***spInstId***, CirId ***circuit***, SiResmEvnt ****resmEvnt***)

Argument	Description
ctahd	Natural Access handle returned by **ctaCreateContext**.
suId	ISUP service access point.
suInstId	Service user instance ID.
spInstId	Service provider instance ID.
circuit	Circuit with which this message is associated.
resmEvnt	Pointer to the caller's resume event structure containing all parameters (IEs) included in the message to the far exchange.

Return values

Return value	Description
ISUP_SUCCESS	
CTAERR_BAD_ARGUMENT	One or more arguments are invalid.
CTAERR_DRIVER_SEND_FAILED	Error occurred accessing the TX driver.
CTAERR_INVALID_CTAHD	Natural Access handle is invalid.

Details

The value specified in the ***suInstId*** field is passed to all subsequent events associated with this connection.

The value for ***spInstId*** must be the ***spInstId*** value that was received from the ISUP layer in the connect indication event.

Example

```
DWORD               status;
SiResmEvnt          resmEvnt;
IsupRcvInfoBlk      rcvInfo;
S16 switchType =    ST_ITUWHITE;
CTAHD               ctahd = Valid CTA Handle;

status = ISUPRetrieveMessage(ctahd, &rcvEvent, &rcvInfo, 1);
if (status == ISUP_NOMSG)
    {
        fprintf(stderr, "ISUPRetrieveMessage() did not get a message, probably a
            congestion event \n");
        return(ISUP_SUCCESS);
    }
if (status != ISUP_SUCCESS)
    {
        fprintf(stderr, "ISUPRetrieveMessage() failed, returned %d\n", status);
        return(status);
    }

...
...

ISUPInitRES( switchType,  &resmEvnt);

status = ISUPResumeReq( ctahd, rcvInfo.suId,  rcvInfo.suInstId,  rcvInfo.spInstId,
             rcvInfo.circuit,  &resmEvnt);

if(status != ISUP_SUCCESS)
    printf("ISUPResumeReq() failed, status = %d\n", status);
else
    printf("RES request sent for circuit %ld\n", circuit);
```

ISUPRetrieveMessage

Retrieves the next message from the ISUP layer.

Prototype

DWORD **ISUPRetrieveMessage** (CTAHD ***ctahd***, SiAllSdus ***event***, IsupRcvInfoBlk ***infoBlk***, Bool ***wait***)

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td>ctahd</td><td>Natural Access handle returned by ctaCreateContext.</td></tr>
<tr><td>event</td><td>Pointer to the address of the caller's event buffer where the received event (if any) is returned to the caller. This buffer must be large enough to accommodate any of the events, as defined by the SiAllSdus structure (union of all event structures). The actual event structure returned (which member of the union) depends on the value of the infoBlk.indType field returned. Refer to the Details section for a list of possible values.</td></tr>
<tr><td>infoBlk</td><td>Pointer to the address of the caller's receive information block where information regarding the received event (if any) is returned to the caller.<pre>typedef struct rcvInfoBlk
{
   U8   indType; /* Ind confirm. type   */
   U8   evntType; /* Event type for status & connection status */
                      /* indications   */
   SuId   suId;   /* Service access point (SAP) id-all   */
   SiInstId suInstId; /* Caller's reference number-all   */
   SiInstId spInstId; /* ISUP's reference number-all   */
   CirId   circuit; /* Circuit id - all   */
   Bool   globalFlg /* Global/circuit specific flag - status ind.*/
                      /* only   */
   U8   spare;   /* Filler for future use   */
} IsupRcvInfoBlk;</pre></td></tr>
<tr><td>wait</td><td>Not used.</td></tr>
</table>

Return values

Return value	Description
ISUP_SUCCESS	
ISUP_NOMSG	No event messages waiting.
ISUP_RESOURCES	Message buffer could not be allocated.

Details

ISUPRetrieveMessage receives events (messages) from the ISUP layer.

When a message is received, **ISUPRetrieveMessage** 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 indication type (indType) identifies the event received and is coded to one of the following values:

EVTSITCNSTIND	0x5A	Connection status indication
EVTSITCONCFM	0x0D	Connect confirm
EVTSITCONIND	0x0E	Connect indication
EVTSITDATIND	0x16	Data indication
EVTSITFACIND	0x6A	Call facility indication
EVTSITRELCFM	0x5D	Connection release confirmation
EVTSITRELIND	0x5E	Connection release indication
EVTSITRESMIND	0x36	Call resume indication
EVTSITSTAIND	0x7A	Status indication
EVTSITSUSPIND	0x3A	Call suspend indication

The following NTT-specific value can be received in the evntType member of the IsupRcvInfoBlk for indType equal to EVTSITCNSTIND:

CHARGE ***Charge Message***

The evntType field identifies the actual message received for connection status, status, and facility indications. It is coded to a value in the following tables:

Connection status indications

Value	Description
ADDRCMPLT	Address complete
FRWDTRSFR	Forward transfer
IDENTREQ	Identification request
IDENTRSP	Identification response
INFORMATION	Information (response to INFORMATREQ)
INFORMATREQ	Information request
MODCMPLT	Call modification complete
MODIFY	Call modification request
MODREJ	Call modification rejected
NETRESMGR	Network resource manager
PROGRESS	Call progress information
SUBSADDR	Subsequent address message

Status indications

Value	Description
CIRBLKREQ	Circuit block request (not supported in BICC)
CIRBLKRSP	Circuit block response (not supported in BICC)
CIRGRPBLKREQ	Circuit group block request
CIRGRPBLKRSP	Circuit group block response
CIRGRPGET	Circuit group get status
CIRGRPQRYRSP	Circuit group query response
CIRGRPRESACK	Circuit group reset acknowledgment
CIRGRPSET	Circuit group set status
CIRGRPUNBLKREQ	Circuit group unblock request (not supported in BICC)
CIRGRPUNBLKRSP	Circuit group unblock response (not supported in BICC)
CIRRESERVE	Circuit reservation request
CIRRESERVEACK	Circuit reservation acknowledgment
CIRRESREQ	Circuit reset request
CIRUNBLKREQ	Circuit unblock request
CIRUNBLKRSP	Circuit unblock response
CIRUNEQPD	Circuit unequipped indication
CONFUSION	Confusion indication
CONTCHK	Continuity check (not supported in BICC)
CONTREP	Continuity report
ERRORIND	Error indication
LOOPBCKACK	Loop back acknowledgment (not supported in BICC)
MTPBACKUP	BACKUP received from MTP
MTPCONGEST	Congestion indication received from MTP
MTPPAUSE	Pause indication received from MTP
MTPPRIMARY	PRIMARY received from MTP
MTPRESUME	Resume indication received from MTP
MTPSTANDALONE	STANDALONE received from MTP
MTPSTOPCONGEST	Stop congestion indication received from MTP
REATTEMPT	Reattempt indication
RMTUSRAVAIL	Remote user available
RMTUSRUNAVAIL	Remote user unavailable
STPCONTIN	Stop continuity indication

Facility indications

Value	Description
FACILITY	Facility
FACILITYACC	Facility accept
FACILITYDEACT	Facility deactivate
FACILITYINFO	Facility information
FACILITYREJ	Facility reject
FACILITYREQ	Facility request

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

The event structure associated with a received message depends on the type of message received from the ISUP layer (as determined by the value of the infoBlk.indType field).

Indication type	Event structure employed
EVTSITCONCFM	SiConEvnt
EVTSITCONIND	SiConEvnt
EVTSITCNSTIND	SiCnStEvnt
EVTSITDATIND	SiInfoEvnt
EVTSITFACCFM	SiFacEvnt
EVTSITFACIND	SiFacEvnt
EVTSITRAWIND	SiRawEvnt
EVTSITRELCFM	SiRelEvnt
EVTSITRELIND	SiRelEvnt
EVTSITRESMIND	SiResmEvnt
EVTSITSTAIND	SiStaEvnt
EVTSITSUSPIND	SiSuspEvnt

Example

```
DWORD status;
SiAllSdus       rcvEvent;
SiAllSdus       sendEvent;
IsupRcvInfoBlk  rcvInfo;

/* Handling all incoming ISUP messages */
status = ISUPRetrieveMessage(ctahd, &rcvEvent, &rcvInfo, 1);
if (status == ISUP_NOMSG)
    {
         fprintf(stderr, "ISUPRetrieveMessage() did not get a message, probably a
             congestion event \n");
         return(ISUP_SUCCESS);
    }
if (status != ISUP_SUCCESS)
    {
         fprintf(stderr, "ISUPRetrieveMessage() failed, returned %d\n", status);
         return(status);
    }

/* determine indication/confirmation type received */
switch( rcvInfo.indType )
{
    ...
    ...
    case EVTSITRELIND:               /* Release indication */
        printf("Release Indication for circuit %ld\n", rcvInfo.circuit);

        status = ISUPReleaseResp(ctahd, rcvInfo.suId, rcvInfo.suInstId,
            rcvInfo.spInstId, rcvInfo.circuit, &rcvEvent.m.siRelEvnt);
        if (status != ISUP_SUCCESS)
            printf("orig: ISUPReleaseResp() failed status = %d\n", status);
        else
            printf("orig: Release Complete sent for circuit %ld\n", rcvInfo.circuit);
            break;

    case EVTSITRELCFM:               /* Release confirmation (release complete) */
        ...
        ...

}
```

ISUPStatusReq

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

Prototype

DWORD **ISUPStatusReq** (CTAHD ***ctahd***, SuId ***suId***, SiInstId ***suInstId***, SiInstId ***spInstId***, Bool ***globalFlg***, CirId ***circuit***, U8 ***eventType***, SiStaEvnt ****statEvnt***)

Argument	Description
ctahd	Natural Access handle returned by **ctaCreateContext**.
suId	ISUP service access point.
suInstId	Service user instance ID.
spInstId	Service provider instance ID.
globalFlg	True (non-zero) if this is a global request; false (zero) if this is a circuit-specific request.
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.
eventType	Type of status request. Refer to the Details section for a list of valid values.
statEvnt	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
ISUP_SUCCESS	
CTAERR_BAD_ARGUMENT	One or more arguments are invalid.
CTAERR_DRIVER_SEND_FAILED	Error occurred accessing the TX driver.
CTAERR_INVALID_CTAHD	Natural Access handle is invalid.

Details

The following table lists the valid status request types for ***eventType***:

Status request type	Description
CONTCHK	Continuity check (not supported in BICC)
CONTREP	Continuity report
LOOPBCKACK	Loop back acknowledgment (not supported in BICC)
CIRRESERVE	Circuit reservation request
CIRRESERVEACK	Circuit reservation acknowledgment
CIRGRPQRYREQ	Circuit group query request
CIRGRPQRYRSP	Circuit group query response
CIRBLKREQ	Circuit block request (not supported in BICC)
CIRBLKRSP	Circuit block response (not supported in BICC)
CIRUNBLKREQ	Circuit unblock request (not supported in BICC)
CIRUNBLKRSP	Circuit unblock response (not supported in BICC)
CIRRESREQ	Circuit reset request
CIRGRPBLKREQ	Circuit group block request
CIRGRPBLKRSP	Circuit group block response
CIRGRPUNBLKREQ	Circuit group unblock request
CIRGRPUNBLKRSP	Circuit group unblock response
CIRGRPRES	Circuit group reset request
CIRGRPSET	Circuit group set request
CIRGRPGET	Circuit group get request

Example

```
#define SAP_ID 0

DWORD       status;
SiStaEvnt   staEvnt;
CTAHD       FstCtaHd = Valid CTA Handle;
SiInstId    suInstId = 0;
SiInstId    spInstId = 0;
SuId        suId = SAP_ID;
CirId       circuit = 2;

memset(&staEvnt,  0,  sizeof(SiStaEvnt));

status = ISUPStatusReq( FstCtaHd,  suId,  suInstId,  spInstId,  0,  circuit,  CIRBLKREQ,
    &staEvnt);

if(status != ISUP_SUCCESS)
    printf("ISUPStatusReq() failed sending BLOck request, status = %d\n", status);
else
    printf("BLOck request sent for circuit %ld\n", circuit);
```

ISUPSuspendReq

Sends a suspend message to the far exchange and starts the suspend timer T2.

Prototype

DWORD **ISUPSuspendReq** (CTAHD ***ctahd***, SuId ***suId***, SiInstId ***suInstId***, SiInstId ***spInstId***, CirId ***circuit***, SiSuspEvnt ****suspEvnt***)

Argument	Description
ctahd	Natural Access handle returned by **ctaCreateContext**.
suId	ISUP service access point.
suInstId	Service user instance ID.
spInstId	Service provider instance ID.
circuit	Circuit with which this message is associated.
suspEvnt	Pointer to the caller's suspend event structure containing all parameters (IEs) to be included in the message to the far exchange.

Return values

Return value	Description
ISUP_SUCCESS	
CTAERR_BAD_ARGUMENT	One or more arguments are invalid.
CTAERR_DRIVER_SEND_FAILED	Error occurred accessing the TX driver.
CTAERR_INVALID_CTAHD	Natural Access handle is invalid.

Details

If the ISUP layer receives no resume or release message before the expiration of the T2 timer, the ISUP layer clears the call in both directions.

The value specified in the ***suInstId*** field is passed to all subsequent events associated with this connection.

The value for ***spInstId*** must be the ***spInstId*** value that was received from the ISUP layer in the connect indication event.

Example

```
DWORD       status;
SiSuspEvnt  suspEvnt;
IsupRcvInfoBlk  rcvInfo;
S16 switchType = ST_ITUWHITE;
CTAHD          ctahd = Valid CTA Handle;

status = ISUPRetrieveMessage(ctahd, &rcvEvent, &rcvInfo, 1);
if (status == ISUP_NOMSG)
    {

       fprintf(stderr, "ISUPRetrieveMessage() did not get a message, probably a congestio
n event \n");
         return(ISUP_SUCCESS);
    }
    if (status != ISUP_SUCCESS)
    {
         fprintf(stderr, "ISUPRetrieveMessage() failed, returned %d\n", status);
         return(status);
    }

...
...

ISUPInitSUS( switchType,  &suspEvnt);

status = ISUPSuspendReq( ctahd, rcvInfo.suId,  rcvInfo.suInstId,  rcvInfo.spInstId,
    rcvInfo.circuit,  &suspEvnt);

if(status != ISUP_SUCCESS)
    printf("ISUPSuspendReq() failed, status = %d\n", status);
else
    printf("SUSpend request sent for circuit %ld\n", circuit);
```

6 Function event initialization routines

Using the function event initialization routines

This section describes the ISUP functions that initialize event structures to generate specific ISUP protocol messages. Each function description contains tables that specify the initialized fields for each supported switch type. The tables are formatted using the following standards:

- Each table row corresponds to an information element or to a field within an information element, and each column corresponds to an ISUP variant.
- Fields are indented under the information element that contains them.
- Required information elements appear in bold face.
- NOT_PRESENT indicates an optional parameter.
- A blank table cell indicates that the parameter is not supported for this variant.

ISUPASCIItoBCD

Converts a string of ASCII digits to a binary-coded decimal string.

Prototype

U8 TXISUPAPIFUNC **ISUPASCIItoBCD** (U8 ***ascii**, U8 ***bcd**, U8 ***length***)

Argument	Description
ascii	Pointer to NULL terminated ASCII string.
bcd	Pointer to destination string.
length	Number of bytes in destination.

Return values

ISUPASCIItoBCD returns the number of successfully converted digits if successful. If unsuccessful, this function returns a zero.

```
void prtTknAddr(TknStr *tk1, char* name)
{
   U8          *called = “8479258900”;
   U8           cdPty[20];
   U8           cdPtyLen = 0;
   U8          *calling = “8479258901”;
   U8           cgPty[20];
   U8           cgPtyLen = 0;
   S16         switchType = ST_ANSI92;
   SiConEvnt iamEvnt;

   cdPtyLen = ISUPASCIItoBCD(called, cdPty, 20);
   cgPtyLen = ISUPASCIItoBCD(calling, cgPty, 20);
   ….
   ….
```

ISUPBCDtoASCII

Converts a binary-coded decimal string to a string of ASCII digits.

Prototype

U8 TXISUPAPIFUNC **ISUPBCDtoASCII** (U8 ***bcd**, U8 ***bcdLen***, U8 ***ascii**, U8 ***asciiLen***)

Argument	Description
bcd	Pointer to binary-coded decimal string.
bcdLen	Number of BCD digits in source string.
ascii	Pointer to destination string.
asciiLen	Number of bytes in destination string. Must be at least ***bcdLen*** plus one byte for NULL termination.

Return values

ISUPBCDtoASCII returns the number of successfully converted digits if successful. If unsuccessful, this function returns a zero.

Example

```
void prtTknAddr(TknStr *tk1, char* name)
{
    U8   addr[64];
    U8   addrLen;

     if(tk1->pres == PRESENT)
     {
        addrLen = tk1->len << 1;
        ISUPBCDtoASCII( tk1->val, addrLen, addr, sizeof( addr ) );
```

ISUPInitACM

Initializes a SiCnStEvnt structure for transmitting an address complete message (ACM).

Prototype

S16 TXISUPAPIFUNC **ISUPInitACM** (S16 ***switchType***, SiCnStEvnt ***event***)

Argument	Description
switchType	One of the following switch type indicators: ST_ANS88 ST_ANS92 ST_ANS95 ST_BICC ST_ETSIV2 ST_ETSIV3 ST_ITU97 ST_ITUBLUE ST_ITUWHITE ST_JNTT ST_Q767
event	Pointer to the SiCnStEvnt structure to be initialized.

Details

The fields of the SiCnStEvnt structure are initialized as described in the following tables, based on the ***switchType*** parameter. Fields not described are not applicable to an ACM. This function is called in preparation for a call to **ISUPConnectStatusReq**.

- ANSI 88, ANSI 92, and ANSI 95 values
- ITU Blue Book, ITU White Book, and ITU Q.767 values
- ITU97, ETSI V2, and ETSI V3 values
- BICC values
- NTT values

ANSI 88, ANSI 92, and ANSI 95 values

Field	ANSI 88	ANSI 92	ANSI 95
bckCallInd	**Present**	**Present**	**Present**
chrgInd	CHRG_NOIND	CHRG_NOIND	CHRG_NOIND
cadPtyStatInd	CADSTAT_NOIND	CADSTAT_NOIND	CADSTAT_NOIND
cadPtyCatInd	CADCAT_ORDSUBS	CADCAT_ORDSUBS	CADCAT_ORDSUBS
end2EndMethInd	E2EMTH_NOMETH	E2EMTH_NOMETH	E2EMTH_NOMETH
intInd	INTIND_NOINTW	INTIND_NOINTW	INTIND_NOINTW
segInd		SEGIND_NOIND	SEGIND_NOIND
end2EndInfoInd	E2EINF_NOINFO	E2EINF_NOINFO	E2EINF_NOINFO
isdnUsrPrtInd	ISUP_USED	ISUP_USED	ISUP_USED
holdInd	HOLD_NOTREQD	HOLD_NOTREQD	HOLD_NOTREQD
isdnAccInd	ISDNACC_ISDN	ISDNACC_ISDN	ISDNACC_ISDN
echoCtrlDevInd		ECHODEV_NOTINCL	ECHODEV_NOTINCL
sccpMethInd		SCCPMTH_NOIND	SCCPMTH_NOIND
accTrnspt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
businessGrp		NOT_PRESENT	NOT_PRESENT
callRef	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
causeDgn		NOT_PRESENT	NOT_PRESENT
connReq	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
infoInd		NOT_PRESENT	NOT_PRESENT
netTransport		NOT_PRESENT	NOT_PRESENT
notifInd		NOT_PRESENT	NOT_PRESENT
optBckCalInd		NOT_PRESENT	NOT_PRESENT
redirInfo		NOT_PRESENT	NOT_PRESENT
remotOper			NOT_PRESENT
serviceAct			NOT_PRESENT
txMedUsed			NOT_PRESENT
usr2UsrInd			NOT_PRESENT
usr2UsrInfo	NOT_PRESENT		NOT_PRESENT

ITU Blue Book, ITU White Book, and ITU Q.767 values

Field	ITU Blue Book	ITU White Book	ITU Q.767
bckCallInd	**Present**	**Present**	**Present**
chrgInd	CHRG_NOIND	CHRG_NOIND	CHRG_NOIND
cadPtyStatInd	CADSTAT_NOIND	CADSTAT_NOIND	CADSTAT_NOIND
cadPtyCatInd	CADCAT_ORDSUBS	CADCAT_ORDSUBS	CADCAT_ORDSUBS
end2EndMethInd	E2EMTH_NOMETH	E2EMTH_NOMETH	E2EMTH_NOMETH
intInd	INTIND_NOINTW	INTIND_NOINTW	INTIND_NOINTW
end2EndInfoInd	E2EINF_NOINFO	E2EINF_NOINFO	E2EINF_NOINFO
isdnUsrPrtInd	ISUP_USED	ISUP_USED	ISUP_USED
holdInd	HOLD_NOTREQD	HOLD_NOTREQD	HOLD_NOTREQD
isdnAccInd	ISDNACC_ISDN	ISDNACC_ISDN	ISDNACC_ISDN
echoCtrlDevInd	ECHODEV_NOTINCL	ECHODEV_NOTINCL	ECHODEV_NOTINCL
sccpMethInd	SCCPMTH_NOIND	SCCPMTH_NOIND	SCCPMTH_NOIND
accDelInfo		NOT_PRESENT	
accTrnspt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
callRef	NOT_PRESENT	NOT_PRESENT	
causeDgn	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
cllDivr		NOT_PRESENT	
connNum	NOT_PRESENT		
echoControl		NOT_PRESENT	
netFac		NOT_PRESENT	
notifInd		NOT_PRESENT	
optBckCalInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
redirNum		NOT_PRESENT	
redirRstr		NOT_PRESENT	
redirInfo	NOT_PRESENT		NOT_PRESENT
remotOper		NOT_PRESENT	
serviceAct		NOT_PRESENT	
txMedUsed		NOT_PRESENT	
usr2UsrInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
usr2UsrInfo	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT

ITU97, ETSI V2, and ETSI V3 values

Field	ITU97	ETSI V2	ETSI V3
bckCallInd	**Present**	**Present**	**Present**
chrgInd	CHRG_NOIND	CHRG_NOIND	CHRG_NOIND
cadPtyStatInd	CADSTAT_NOIND	CADSTAT_NOIND	CADSTAT_NOIND
cadPtyCatInd	CADCAT_ORDSUBS	CADCAT_ORDSUBS	CADCAT_ORDSUBS
end2EndMethInd	E2EMTH_NOMETH	E2EMTH_NOMETH	E2EMTH_NOMETH
intInd	INTIND_NOINTW	INTIND_NOINTW	INTIND_NOINTW
end2EndInfoInd	E2EINF_NOINFO	E2EINF_NOINFO	E2EINF_NOINFO
isdnUsrPrtInd	ISUP_USED	ISUP_USED	ISUP_USED
holdInd	HOLD_NOTREQD	HOLD_NOTREQD	HOLD_NOTREQD
isdnAccInd	ISDNACC_ISDN	ISDNACC_ISDN	ISDNACC_ISDN
echoCtrlDevInd	ECHODEV_NOTINCL	ECHODEV_NOTINCL	ECHODEV_NOTINCL
sccpMethInd	SCCPMTH_NOIND	SCCPMTH_NOIND	SCCPMTH_NOIND
accDelInfo	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
accTrnspt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
callRef	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
causeDgn	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
cllDivr	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
connNum			
echoControl	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
netFac	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
notifInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
optBckCalInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
redirNum	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
redirRstr	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
redirInfo			
remotOper	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
serviceAct	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
txMedUsed	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
usr2UsrInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
usr2UsrInfo	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
confTrtmnt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
uIDActionInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
cCNR	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT

BICC values

Field	BICC
bckCallInd	**Present**
chrgInd	CHRG_NOIND
cadPtyStatInd	CADSTAT_NOIND
cadPtyCatInd	CADCAT_ORDSUBS
end2EndMethInd	E2EMTH_NOMETH
intInd	INTIND_NOINTW
end2EndInfoInd	E2EINF_NOINFO
isdnUsrPrtInd	BICC_USED
holdInd	HOLD_NOTREQD
isdnAccInd	ISDNACC_ISDN
echoCtrlDevInd	ECHODEV_NOTINCL
sccpMethInd	SCCPMTH_NOIND
accDelInfo	NOT_PRESENT
accTrnspt	NOT_PRESENT
callRef	NOT_PRESENT
causeDgn	NOT_PRESENT
cllDivr	NOT_PRESENT
connNum	
echoControl	NOT_PRESENT
netFac	NOT_PRESENT
notifInd	NOT_PRESENT
optBckCalInd	NOT_PRESENT
redirNum	NOT_PRESENT
redirRstr	NOT_PRESENT
redirInfo	
remotOper	NOT_PRESENT
serviceAct	NOT_PRESENT
txMedUsed	NOT_PRESENT
usr2UsrInd	NOT_PRESENT
usr2UsrInfo	NOT_PRESENT

NTT values

Field	NTT
bckCallInd	**Present**
chrgInd	CHRG_NOIND
cadPtyStatInd	CADSTAT_NOIND
cadPtyCatInd	CADCAT_ORDSUBS
end2EndMethInd	E2EMTH_NOMETH
intInd	INTIND_NOINTW
end2EndInfoInd	E2EINF_NOINFO
isdnUsrPrtInd	ISUP_USED
holdInd	HOLD_NOTREQD
isdnAccInd	ISDNACC_ISDN
echoCtrlDevInd	ECHODEV_NOTINCL
sccpMethInd	SCCPMTH_NOIND
accTrnspt	NOT_PRESENT
causeDgn	NOT_PRESENT
optBckCalInd	NOT_PRESENT
usr2UsrInfo	NOT_PRESENT
msgAreaInfo	NOT_PRESENT
chargeInfo	NOT_PRESENT
chargeInfoType	NOT_PRESENT
chargeInfoDly	NOT_PRESENT
carrierInfoTrans	NOT_PRESENT

Example

```
S16        switchType=ST_ITU97;
SiAllSdus  sendEvent;

ISUPInitACM(switchType, &sendEvent.m.SiCnStEvnt);
```

ISUPInitANM

Initializes a SiConEvnt structure for transmitting an answer message (ANM).

Prototype

S16 TXISUPAPIFUNC **ISUPInitANM** (S16 ***switchType***, SiConEvnt ***event***)

	Description
switchType	One of the following switch type indicators: ST_ANS92 ST_ANS95 ST_BICC ST_ETSIV2 ST_ITU97 ST_ITUBLUE ST_ITUWHITE ST_JNTT
event	Pointer to the SiConEvnt structure to be initialized.

Details

The fields of the SiConEvnt structure are initialized as described in the following tables, based on the ***switchType*** parameter. Fields not described are not applicable to an ANM. This function is called in preparation for a call to **ISUPConnectResp**.

- ANSI 88, ANSI 92, and ANSI 95 values
- ITU Blue Book, ITU White Book, and ITU Q.767 values
- ITU 97, ETSI V2, and ETSI V3 values
- BICC values
- NTT values

ANSI 88, ANSI 92, and ANSI 95 values

Field	ANSI 88	ANSI 92	ANSI 95
accTrnspt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
bckCallInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
businessGrp		NOT_PRESENT	NOT_PRESENT
callRef	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
connReq	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
infoInd		NOT_PRESENT	NOT_PRESENT
netTransport		NOT_PRESENT	NOT_PRESENT
notifInd		NOT_PRESENT	NOT_PRESENT
optBckCalInd		NOT_PRESENT	NOT_PRESENT
remotOper			NOT_PRESENT
serviceAct			NOT_PRESENT
txMedUsed			NOT_PRESENT
usr2UsrInd			NOT_PRESENT
usr2UsrInfo	NOT_PRESENT		NOT_PRESENT

ITU Blue Book, ITU White Book, and ITU Q.767 values

Field	ITU Blue Book	ITU White Book	ITU Q.767
accTrnspt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
accDelInfo		NOT_PRESENT	
bckCallInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
callRef	NOT_PRESENT	NOT_PRESENT	
cllHstry		NOT_PRESENT	
connReq			
connNum	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
genNmb		NOT_PRESENT	
netFac		NOT_PRESENT	
notifInd		NOT_PRESENT	
optBckCalInd	NOT_PRESENT	NOT_PRESENT	
parmCom		NOT_PRESENT	
redirNum		NOT_PRESENT	
redirRstr		NOT_PRESENT	
remotOper		NOT_PRESENT	
serviceAct		NOT_PRESENT	
txMedUsed		NOT_PRESENT	
usr2UsrInd	NOT_PRESENT	NOT_PRESENT	
usr2UsrInfo	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT

ITU 97, ETSI V2, and ETSI V3 values

Field	ITU97	ETSI V2	ETSI V3
accTrnspt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
accDelInfo	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
bckCallInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
callRef	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
cllHstry	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
connReq			
connNum	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
genNmb	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
netFac	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
notifInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
optBckCalInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
parmCom	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
redirNum	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
redirRstr	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
remotOper	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
serviceAct	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
txMedUsed	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
usr2UsrInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
usr2UsrInfo	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
bckGVNS	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT

BICC values

Field	BICC
accTrnspt	NOT_PRESENT
accDelInfo	NOT_PRESENT
bckCallInd	NOT_PRESENT
callRef	NOT_PRESENT
cllHstry	NOT_PRESENT
connReq	
connNum	NOT_PRESENT
genNmb	NOT_PRESENT
netFac	NOT_PRESENT
notifInd	NOT_PRESENT
optBckCalInd	NOT_PRESENT
parmCom	NOT_PRESENT
redirNum	NOT_PRESENT
redirRstr	NOT_PRESENT
remotOper	NOT_PRESENT
serviceAct	NOT_PRESENT
txMedUsed	NOT_PRESENT
usr2UsrInd	NOT_PRESENT
usr2UsrInfo	NOT_PRESENT

NTT values

Field	NTT
accTrnspt	NOT_PRESENT
bckCallInd	NOT_PRESENT
msgAreaInfo	NOT_PRESENT
usr2UsrInfo	NOT_PRESENT

Example

```
S16         switchType=ST_ITU97;
SiAllSdus   sendEvent;

ISUPInitANM(switchType, &sendEvent.m.SiConEvnt);
```

ISUPInitCON

Initializes a SiConEvnt structure for transmitting a connect message (CON).

Prototype

S16 TXISUPAPIFUNC **ISUPInitCON** (S16 ***switchType***, SiConEvnt ***event**)

Argument	Description
switchType	One of the following switch type indicators: ST_BICC ST_ETSIV2 ST_ETSIV3 ST_ITU97 ST_ITUBLUE ST_ITUWHITE ST_Q767
event	Pointer to the SiConEvnt structure to be initialized.

Details

The fields of the SiConEvnt structure are initialized as described in the following tables, based on the ***switchType*** parameter. Fields not described are not applicable to a CON. This function is called in preparation for a call to **ISUPConnectResp**.

- ITU Blue Book, ITU White Book, and ITU Q.767 values
- ITU 97, ETSI V2, and ETSI V3 values
- BICC values

ITU Blue Book, ITU White Book, and ITU Q.767 values

Field	ITU Blue Book	ITU White Book	ITU Q.767
bckCallInd	**Present**	**Present**	**Present**
chrgInd	CHRG_NOIND	CHRG_NOIND	CHRG_NOIND
cadPtyStatInd	CADSTAT_NOIND	CADSTAT_NOIND	CADSTAT_NOIND
cadPtyCatInd	CADCAT_ORDSUBS	CADCAT_ORDSUBS	CADCAT_ORDSUBS
end2EndMethInd	E2EMTH_NOMETH	E2EMTH_NOMETH	E2EMTH_NOMETH
intInd	INTIND_NOINTW	INTIND_NOINTW	INTIND_NOINTW
end2EndInfoInd	E2EINF_NOINFO	E2EINF_NOINFO	E2EINF_NOINFO
isdnUsrPrtInd	ISUP_USED	ISUP_USED	ISUP_USED
holdInd	HOLD_NOTREQD	HOLD_NOTREQD	HOLD_NOTREQD
isdnAccInd	ISDNACC_ISDN	ISDNACC_ISDN	ISDNACC_ISDN
echoCtrlDevInd	ECHODEV_NOTINCL	ECHODEV_NOTINCL	ECHODEV_NOTINCL
sccpMethInd	SCCPMTH_NOIND	SCCPMTH_NOIND	SCCPMTH_NOIND
optBckCalInd	NOT_PRESENT	NOT_PRESENT	
connNum	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
callRef	NOT_PRESENT	NOT_PRESENT	
accTrnspt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
accDelInfo		NOT_PRESENT	
netFac		NOT_PRESENT	
cllHstry		NOT_PRESENT	
genNmb		NOT_PRESENT	
causeDgn			
connReq			
echoControl		NOT_PRESENT	
notifInd		NOT_PRESENT	
parmCom		NOT_PRESENT	
redirNum		NOT_PRESENT	
redirRstr		NOT_PRESENT	
redirInfo			
remotOper		NOT_PRESENT	
serviceAct		NOT_PRESENT	
txMedUsed		NOT_PRESENT	
usr2UsrInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
usr2UsrInfo	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT

ITU 97, ETSI V2, and ETSI V3 values

Field	ITU97	ETSI V2	ETSI V3
bckCallInd	**Present**	**Present**	**Present**
chrgInd	CHRG_NOIND	CHRG_NOIND	CHRG_NOIND
cadPtyStatInd	CADSTAT_NOIND	CADSTAT_NOIND	CADSTAT_NOIND
cadPtyCatInd	CADCAT_ORDSUBS	CADCAT_ORDSUBS	CADCAT_ORDSUBS
end2EndMethInd	E2EMTH_NOMETH	E2EMTH_NOMETH	E2EMTH_NOMETH
intInd	INTIND_NOINTW	INTIND_NOINTW	INTIND_NOINTW
end2EndInfoInd	E2EINF_NOINFO	E2EINF_NOINFO	E2EINF_NOINFO
isdnUsrPrtInd	ISUP_USED	ISUP_USED	ISUP_USED
holdInd	HOLD_NOTREQD	HOLD_NOTREQD	HOLD_NOTREQD
isdnAccInd	ISDNACC_ISDN	ISDNACC_ISDN	ISDNACC_ISDN
echoCtrlDevInd	ECHODEV_NOTINCL	ECHODEV_NOTINCL	ECHODEV_NOTINCL
sccpMethInd	SCCPMTH_NOIND	SCCPMTH_NOIND	SCCPMTH_NOIND
optBckCalInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
connNum	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
callRef	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
accTrnspt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
accDelInfo	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
netFac	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
cllHstry	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
genNmb	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
causeDgn			
connReq			
echoControl	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
notifInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
parmCom	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
redirNum	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
redirRstr	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
redirInfo			
remotOper	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
serviceAct	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
txMedUsed	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
usr2UsrInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
usr2UsrInfo	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
bckGVNS	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
confTrtmnt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT

BICC values

Field	BICC
bckCallInd	**Present**
chrgInd	CHRG_NOIND
cadPtyStatInd	CADSTAT_NOIND
cadPtyCatInd	CADCAT_ORDSUBS
end2EndMethInd	E2EMTH_NOMETH
intInd	INTIND_NOINTW
end2EndInfoInd	E2EINF_NOINFO
isdnUsrPrtInd	BICC_USED
holdInd	HOLD_NOTREQD
isdnAccInd	ISDNACC_ISDN
echoCtrlDevInd	ECHODEV_NOTINCL
sccpMethInd	SCCPMTH_NOIND
optBckCalInd	NOT_PRESENT
connNum	NOT_PRESENT
callRef	NOT_PRESENT
accTrnspt	NOT_PRESENT
accDelInfo	NOT_PRESENT
netFac	NOT_PRESENT
cllHstry	NOT_PRESENT
genNmb	NOT_PRESENT
causeDgn	
connReq	
echoControl	NOT_PRESENT
notifInd	NOT_PRESENT
parmCom	NOT_PRESENT
redirNum	NOT_PRESENT
redirRstr	NOT_PRESENT
redirInfo	
remotOper	NOT_PRESENT
serviceAct	NOT_PRESENT
txMedUsed	NOT_PRESENT
usr2UsrInd	NOT_PRESENT
usr2UsrInfo	NOT_PRESENT

Example

```
S16         switchType=ST_ITUWHITE;
SiAllSdus   sendEvent;

ISUPInitCON(switchType, &sendEvent.m.SiConEvnt);
```

ISUPInitCOT

Initializes a SiStaEvnt structure for transmitting a continuity message (COT).

Prototype

S16 TXISUPAPIFUNC **ISUPInitCOT** (S16 ***switchType***, SiStaEvnt ***event**, U8 ***contInd***)

Argument	Description
switchType	One of the following switch type indicators: ST_ANS88 ST_ANS92 ST_ANS95 ST_BICC ST_ITUBLUE ST_ITUWHITE ST_Q767
event	Pointer to the SiStaEvnt structure to be initialized.
contInd	Continuity indicator. Must be either CONT_CHKFAIL or CONT_CHKSUCC.

Details

The fields of the SiStaEvnt structure are initialized as described in the following tables, based on the ***switchType*** parameter. Fields not described are not applicable to a COT. This function is called in preparation for a call to **ISUPStatusReq**.

- ANSI 88, ANSI 92, and ANSI 95 values
- ITU Blue Book, ITU White Book, and ITU Q.767 values
- BICC values

ANSI 88, ANSI 92, and ANSI 95 values

Field	ANSI 88	ANSI 92	ANSI 95
contInd	**from contInd**	**from contInd**	**from contInd**

ITU Blue Book, ITU White Book, and ITU Q.767 values

Field	ITU Blue Book	ITU White Book	ITU Q.767
contInd	**from contInd**	**from contInd**	**from contInd**

BICC values

Field	BICC
contInd	**from contInd**

Example

```
S16        switchType=ST_ANS88;
SiAllSdus  sendEvent;
U8         contInd=CONT_CHKSUCC;

ISUPInitCOT(switchType, &sendEvent.m.SiStaEvnt, contInd);
```

ISUPInitCPG

Initializes a SiCnStEvnt structure for transmitting a call progress message (CPG).

Prototype

S16 TXISUPAPIFUNC **ISUPInitCPG** (S16 ***switchType***, SiCnStEvnt ***event**)

Argument	Description
switchType	One of the following switch type indicators: ST_ANS92 ST_ANS95 ST_BICC ST_ETSIV2 ST_ETSIV3 ST_ITU97 ST_ITUBLUE ST_ITUWHITE ST_JNTT ST_Q767
event	Pointer to the SiCnStEvnt structure to be initialized.

Details

The fields of the SiCnStEvnt structure are initialized as described in the following tables, based on the ***switchType*** parameter. Fields not described are not applicable to a CPG. This function is called in preparation for a call to **ISUPConnectStatusReq**.

- ANSI 92 and ANSI 95 values
- ITU Blue Book, ITU White Book, and ITU Q.767 values
- ITU 97, ETSI V2, and ETSI V3 values
- BICC values
- NTT values

ANSI 92 and ANSI 95 values

Field	ANSI 92	ANSI 95
evntInfo	**Present**	**Present**
evntInd	EV_PROGRESS	EV_PROGRESS
evtPresResInd	EVPR_NOIND	EVPR_NOIND
accTrnspt	NOT_PRESENT	NOT_PRESENT
bckCallInd	NOT_PRESENT	NOT_PRESENT
businessGrp	NOT_PRESENT	NOT_PRESENT
callRef	NOT_PRESENT	NOT_PRESENT
causeDgn	NOT_PRESENT	NOT_PRESENT
infoInd	NOT_PRESENT	NOT_PRESENT
netTransport	NOT_PRESENT	NOT_PRESENT
notifInd	NOT_PRESENT	NOT_PRESENT
optBckCalInd	NOT_PRESENT	NOT_PRESENT
redirNum		NOT_PRESENT
remotOper		NOT_PRESENT
serviceAct		NOT_PRESENT
txMedUsed		NOT_PRESENT
usr2UsrInd		NOT_PRESENT
usr2UsrInfo		NOT_PRESENT

ITU Blue Book, ITU White Book, and ITU Q.767 values

Field	ITU Blue Book	ITU White Book	ITU Q.767
evntInfo	**Present**	**Present**	**Present**
evntInd	EV_PROGRESS	EV_PROGRESS	EV_PROGRESS
evtPresResInd	EVPR_NOIND	EVPR_NOIND	EVPR_NOIND
accDelInfo		NOT_PRESENT	
accTrnspt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
bckCallInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
callRef	NOT_PRESENT	NOT_PRESENT	
causeDgn	NOT_PRESENT	NOT_PRESENT	
cllDivr		NOT_PRESENT	
notifInd		NOT_PRESENT	
netFac		NOT_PRESENT	
optBckCalInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
parmCom		NOT_PRESENT	
redirNum	NOT_PRESENT	NOT_PRESENT	
redirRstr		NOT_PRESENT	
remotOper		NOT_PRESENT	
serviceAct		NOT_PRESENT	
txMedUsed		NOT_PRESENT	
usr2UsrInd	NOT_PRESENT	NOT_PRESENT	
usr2UsrInfo	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT

ITU 97, ETSI V2, and ETSI V3 values

Field	ITU 97	ETSI V2	ETSI V3
evntInfo	**Present**	**Present**	**Present**
evntInd	EV_PROGRESS	EV_PROGRESS	EV_PROGRESS
evtPresResInd	EVPR_NOIND	EVPR_NOIND	EVPR_NOIND
accDelInfo	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
accTrnspt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
bckCallInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
callRef	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
causeDgn	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
cllDivr	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
notifInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
netFac	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
optBckCalInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
parmCom	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
redirNum	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
redirRstr	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
remotOper	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
serviceAct	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
txMedUsed	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
usr2UsrInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
usr2UsrInfo	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
callXferNum	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
bckGVNS	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
confTrtmnt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
uIDActionInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
cCNR	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT

BICC values

Field	BICC
evntInfo	**Present**
evntInd	EV_PROGRESS
evtPresResInd	EVPR_NOIND
accDelInfo	NOT_PRESENT
accTrnspt	NOT_PRESENT
bckCallInd	NOT_PRESENT
callRef	NOT_PRESENT
causeDgn	NOT_PRESENT
cllDivr	NOT_PRESENT
notifInd	NOT_PRESENT
netFac	NOT_PRESENT
optBckCalInd	NOT_PRESENT
parmCom	NOT_PRESENT
redirNum	NOT_PRESENT
redirRstr	NOT_PRESENT
remotOper	NOT_PRESENT
serviceAct	NOT_PRESENT
txMedUsed	NOT_PRESENT
usr2UsrInd	NOT_PRESENT
usr2UsrInfo	NOT_PRESENT

NTT values

Field	NTT
evntInfo	**Present**
evntInd	EV_PROGRESS
evtPresResInd	EVPR_NOIND
accTrnspt	NOT_PRESENT
bckCallInd	NOT_PRESENT
causeDgn	NOT_PRESENT
serviceAct	
usr2UsrInfo	NOT_PRESENT

Example

```
S16         switchType=ETSIV3;
SiAllSdus   sendEvent;

ISUPInitCPG(switchType, &sendEvent.m.SiCnStEvnt);
```

ISUPInitCRM

Initializes a SiStaEvnt structure for transmitting a circuit reservation message (CRM).

Prototype

S16 TXISUPAPIFUNC **ISUPInitCRM** (S16 ***switchType***, SiStaEvnt ***event***)

Argument	Description
switchType	One of the following switch type indicators: ST_ANS92 ST_ANS95
event	Pointer to the SiStaEvnt structure to be initialized.

Details

The fields of the SiStaEvnt structure are initialized as described in the following table, based on the ***switchType*** parameter. Fields not described are not applicable to a CRM. This function is called in preparation for a call to **ISUPStatusReq**.

Field	ANSI 92	ANSI 95
natConInd	**Present**	**Present**
satInd	SAT_NONE	SAT_NONE
contChkInd	CONTCHK_NOTREQ	CONTCHK_NOTREQ
echoCntrlDevInd	ECHOCDEV_NOTINCL	ECHOCDEV_NOTINCL

Example

```
S16        switchType=ST_ANS95;
SiAllSdus  sendEvent;

ISUPInitCRM(switchType, &sendEvent.m.SiStaEvnt);
```

ISUPInitFAA

Initializes a SiFacEvnt structure for transmitting a facility-accepted message (FAA).

Prototype

S16 TXISUPAPIFUNC **ISUPInitFAA** (S16 ***switchType***, SiFacEvnt ***event***)

Argument	Description
switchType	One of the following switch type indicators: ST_ANS88 ST_BICC ST_ITUBLUE ST_ITUWHITE
event	Pointer to the SiFacEvnt structure to be initialized.

Details

The fields of the SiFacEvnt structure are initialized as described in the following tables, based on the ***switchType*** parameter. Fields not described in the tables are not applicable to an FAA.

- ANSI 88
- ITU Blue Book, ITU White Book, and BICC values

ANSI 88

Field	ANSI 88
facInd	**FI_BUSYFREE**
cdPtyNum	NOT_PRESENT
cgPtyNum	NOT_PRESENT
callRef	NOT_PRESENT

ITU Blue Book, ITU White Book, and BICC values

Field	ITU Blue Book	ITU White Book	BICC
facInd	**FI_USR2USRSERV**	**FI_USR2USRSERV**	**FI_USR2USRSERV**
parmCom		NOT_PRESENT	NOT_PRESENT
usr2UsrInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
callRef	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
connReq		NOT_PRESENT	NOT_PRESENT

Example

```
S16        switchType=ST_ITUBLUE;
SiAllSdus  sendEvent;

ISUPInitFAA(switchType, &sendEvent.m.SiFacEvnt);
```

ISUPInitFAC

Initializes a SiFacEvnt structure for transmitting a facility message (FAC).

Prototype

S16 TXISUPAPIFUNC **ISUPInitFAC** (S16 ***switchType***, SiFacEvnt ***event**)

Argument	Description
switchType	One of the following switch type indicators: ST_ANS95 ST_BICC ST_ETSIV2 ST_ETSIV3 ST_ITU97 ST_ITUWHITE
event	Pointer to the SiFacEvnt structure to be initialized.

Details

The fields in the SiFacEvnt structure are initialized as described in the following tables, based on the ***switchType*** parameter. Fields not described are not applicable to an FAC. This function is called in preparation for a call to **ISUPFacilityReq**.

- ITU White Book and BICC values
- ITU 97, ETSI V2, and ETSI V3 values
- ANSI 95 values

ITU White Book and BICC values

Field	ITU White Book	BICC
facInd	NOT_PRESENT	NOT_PRESENT
remotOper	NOT_PRESENT	NOT_PRESENT
serviceAct	NOT_PRESENT	NOT_PRESENT
msgCom	NOT_PRESENT	NOT_PRESENT
parmCom	NOT_PRESENT	NOT_PRESENT

ITU 97, ETSI V2, and ETSI V3 values

Field	ITU 97	ETSI V2	ETSI V3
msgCom	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
parmCom	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
remotOper	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
callXferNum	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
accTrans	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
notifInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT

ANSI 95 values

Field	ANSI 95
remotOper	NOT_PRESENT
serviceAct	NOT_PRESENT

Example

```
S16         switchType=ST_ITU97;
SiAllSdus   sendEvent;

ISUPInitFAC(switchType, &sendEvent.m.SiFacEvnt);
```

ISUPInitFAD

Initializes a SiFacEvnt structure for transmitting a facility deactivate message (FAD).

Prototype

S16 TXISUPAPIFUNC **ISUPInitFAD** (S16 ***switchType***, SiFacEvnt ***event***)

Argument	Description
switchType	Switch type indicator that must be set to ST_ANS88.
event	Pointer to the SiFacEvnt structure to be initialized.

Details

The fields in the SiFacEvnt structure are initialized as described in the following table, based on the ***switchType*** parameter. Fields not described are not applicable to an FAD. This function is called in preparation for a call to **ISUPFacilityReq**.

Field	ANSI 88
facInd	**FI_BUSYFREE**
cdPtyNum	NOT_PRESENT
cgPtyNum	NOT_PRESENT
callRef	NOT_PRESENT

Example

```
S16        switchType=ST_ANS88;
SiAllSdus  sendEvent;

ISUPInitFAD(switchType, &sendEvent.m.SiFacEvnt);
```

ISUPInitFAI

Initializes a SiFacEvnt structure for transmitting a facility information message (FAI).

Prototype

S16 TXISUPAPIFUNC **ISUPInitFAI** (S16 ***switchType***, SiFacEvnt ***event**)

Argument	Description
switchType	Switch type indicator that must be set to ST_ANS88.
event	Pointer to the SiFacEvnt structure to be initialized.

Details

The fields in the SiFacEvnt structure are initialized as described in the following table, based on the ***switchType*** parameter. Fields not described are not applicable to an FAI. This function is called in preparation for a call to **ISUPFacilityReq**.

Field	ANSI 88
facInd	**FI_BUSYFREE**
facInfInd	
calldPtyFreeInd	CDPTY_FREE
callgPtyAnsInd	NOCGPTYANS
facReqEnqInd	NOENQUIRY
facReqActInd	FACREQNOTACTIVE
cdPtyNum	NOT_PRESENT
cgPtyNum	NOT_PRESENT
callRef	NOT_PRESENT

Example

```
S16        switchType=ST_ANS88;
SiAllSdus  sendEvent;

ISUPInitFAI(switchType, &sendEvent.m.SiFacEvnt);
```

ISUPInitFAR

Initializes a SiFacEvnt structure for transmitting a facility request message (FAR).

Prototype

S16 TXISUPAPIFUNC **ISUPInitFAR** (S16 ***switchType***, SiFacEvnt ***event**)

Argument	Description
switchType	One of the following switch type indicators: ST_ANS88 ST_BICC ST_ITUBLUE ST_ITUWHITE
event	Pointer to the SiFacEvnt structure to be initialized.

Details

The fields of the SiFacEvnt structure are initialized as described in the following tables, based on the ***switchType*** parameter. Fields not described are not applicable to an FAR. This function is called in preparation for a call to **ISUPFacilityReq**.

- ANSI 88 values
- ITU Blue Book, ITU White Book, and BICC values

ANSI 88 values

Field	ANSI 88
facInd	**FI_BUSYFREE**
cdPtyNum	NOT_PRESENT
cgPtyNum	NOT_PRESENT
callRef	NOT_PRESENT

ITU Blue Book, ITU White Book, and BICC values

Field	ITU Blue Book	ITU White Book	BICC
facInd	**FI_USR2USRSERV**	**FI_USR2USRSERV**	**FI_USR2USRSERV**
parmCom		NOT_PRESENT	NOT_PRESENT
usr2UsrInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
callRef	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
connReq		NOT_PRESENT	NOT_PRESENT

Example

```
S16         switchType=ST_ITUWHITE;
SiAllSdus   sendEvent;

ISUPInitFAR(switchType, &sendEvent.m.SiFacEvnt);
```

ISUPInitFOT

Initializes a SiCnStEvnt structure for transmitting a forward transfer message (FOT).

Prototype

S16 TXISUPAPIFUNC **ISUPInitFOT** (S16 ***switchType***, SiCnStEvnt ***event***)

Argument	Description
switchType	One of the following switch type indicators: ST_ANS88 ST_ANS92 ST_ANS95 ST_BICC ST_ITUBLUE ST_ITUWHITE ST_Q767
event	Pointer to the SiCnStEvnt structure to be initialized.

Details

The fields of the SiCnStEvnt structure are initialized as described in the following tables, based on the ***switchType*** parameter. Fields not described are not applicable to an FOT. This function is called in preparation for a call to **ISUPConnectStatusReq**.

- ANSI 88, ANSI 92, and ANSI 95 values
- ITU Blue Book, ITU White Book, ITU Q.767, and BICC values

ANSI 88, ANSI 92, and ANSI 95 values

Field	ANSI 88	ANSI 92	ANSI 95
callRef	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT

ITU Blue Book, ITU White Book, and ITU Q.767, and BICC values

Field	ITU Blue Book	ITU White Book	ITU Q.767	BICC
callRef	NOT_PRESENT	NOT_PRESENT		NOT_PRESENT

Example

```
S16         switchType=ST_ANS95;
SiAllSdus   sendEvent;

ISUPInitFOT(switchType, &sendEvent.m.SiCnStEvnt);
```

ISUPInitFRJ

Initializes a SiFacEvnt structure for transmitting a facility-rejected message (FRJ).

Prototype

S16 TXISUPAPIFUNC **ISUPInitFRJ** (S16 ***switchType***, SiFacEvnt ****event***, U8 ***cause***)

Argument	Description
switchType	One of the following switch type indicators: ST_ANS88 ST_BICC ST_ITUBLUE ST_ITUWHITE
event	Pointer to the SiFacEvnt structure to be initialized.
cause	Cause value.

Details

The fields of the SiFacEvnt structure are initialized as described in the following tables, based on the ***switchType*** parameter. Fields not described are not applicable to an FRJ.

- ANSI 88 values
- ITU Blue Book, ITU White Book, and BICC values

ANSI 88 values

Field	ANSI 88
facInd	**FI_BUSYFREE**
causeDgn	**Present**
location	ILOC_PRIVNETLU
cdeStand	CSTD_NAT
causeVal	from cause
dgnVal	NOT_PRESENT
cdPtyNum	NOT_PRESENT
cgPtyNum	NOT_PRESENT
callRef	NOT_PRESENT

ITU Blue Book, ITU White Book, and BICC values

Field	ITU Blue Book	ITU White Book	BICC
facInd	**FI_USR2USRSERV**	**FI_USR2USRSERV**	**FI_USR2USRSERV**
causeDgn	**Present**	**Present**	**Present**
location	ILOC_PRIVNETLU	ILOC_PRIVNETLU	ILOC_PRIVNETLU
cdeStand	CSTD_NAT	CSTD_NAT	CSTD_NAT
causeVal	from cause	from cause	from cause
dgnVal	NOT_PRESENT	NOT_Present	NOT_PRESENT
callRef	NOT_PRESENT		
usr2UsrInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT

Example

```
S16         switchType=ST_ITUBLUE;
SiAllSdus   sendEvent;
U8          cause= CCFACREJ    /* Decimal 29, defined in iedefs.h */;

ISUPInitFRJ(switchType, &sendEvent.m.SiFacEvnt, cause);
```

ISUPInitIAM

Initializes a SiConEvnt structure for transmitting an initial address message (IAM).

Prototype

S16 TXISUPAPIFUNC **ISUPInitIAM** (S16 ***switchType***, SiConEvnt ****event***, U8 ****cdPty***, U8 ***cdPtyLen***, U8 ****cgPty***, U8 ***cgPtyLen***)

Argument	Description
switchType	One of the following switch type indicators: ST_ANS88 ST_ANS92 ST_ANS95 ST_BICC ST_ETSIV2 ST_ETSIV3 ST_ITU97 ST_ITUBLUE ST_ITUWHITE ST_JNTT ST_Q767
event	Pointer to the SiConEvnt structure to be initialized.
cdPty	Pointer to BCD called party address.
cdPtyLen	Number of BCD digits in called party address.
cgPty	Pointer to BCD calling party address. A null pointer can be passed in this argument.
cgPtyLen	Number of BCD digits in calling party address.

Details

The fields of the SiConEvnt structure are initialized as described in the following tables, based on the ***switchType*** parameter. Fields not described are not applicable to an IAM. This function is called in preparation for a call to **ISUPConnectReq**.

- ANSI 88, ANSI 92, and ANSI 95 values
- ITU Blue Book, ITU White Book, and ITU Q.767 values
- ITU97, ETSI V2, and ETSI V3 values
- BICC values
- NTT values

ANSI 88, ANSI 92, and ANSI 95 values

Field	ANSI 88	ANSI 92	ANSI 95
natConInd	**Present**	**Present**	**Present**
satInd	SAT_NONE	SAT_NONE	SAT_NONE
contChkInd	CONTCHK_NOTREQ	CONTCHK_NOTREQ	CONTCHK_NOTREQ
echoCntrlDevInd	ECHOCDEV_NOTINCL	ECHOCDEV_NOTINCL	ECHOCDEV_NOTINCL
fwdCallInd	**Present**	**Present**	**Present**
natIntCallInd	CALL_NAT	CALL_NAT	CALL_NAT

Field	**ANSI 88**	**ANSI 92**	**ANSI 95**
end2EndMethInd	E2EMTH_NOMETH	E2EMTH_NOMETH	E2EMTH_NOMETH
intInd	INTIND_NOINTW	INTIND_NOINTW	INTIND_NOINTW
intend2EndInfoInd	E2EINF_NOINFO		
segInd		SEGIND_NOINTW	SEGIND_NOINTW
isdnUsrPrtInd	ISUP_USED	ISUP_USED	ISUP_USED
isdnUsrPrtPrfInd	PREF_PREFAW	PREF_PREFAW	PREF_PREFAW
isdnAccInd	ISDNACC_ISDN	ISDNACC_ISDN	ISDNACC_ISDN
sccpMethInd		SCCPMTH_NOIND	SCCPMTH_NOIND
cdPtyNum	**Present**	**Present**	**Present**
natAddrInd	NATNUM	NATNUM	NATNUM
numPlan	NP_ISDN	NP_ISDN	NP_ISDN
innInd	INN_ALLOW	INN_ALLOW	INN_ALLOW
addrSig	from cdPty	from cdPty	from cdPty
oddEven	from cdPtyLen	from cdPtyLen	from cdPtyLen
CgPtyCat	**CAT_ORD**	**CAT_ORD**	**CAT_ORD**
usrServInfo	**Present**	**Present**	**Present**
infoTranCap	ITC_SPEECH	ITC_SPEECH	ITC_SPEECH
cdeStand	CSTD_NAT	CSTD_NAT	CSTD_NAT
infoTranRate0	ITR_64KBIT	ITR_64KBIT	ITR_64KBIT
tranMode	TM_CIRCUIT	TM_CIRCUIT	TM_CIRCUIT
establish	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
config	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
chanStruct	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
infoTranRate1	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
symmetry	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
usrInfLyr1Prot	UIL1_G711ULAW	UIL1_G711ULAW	UIL1_G711ULAW
lyr1Ident	L1_IDENT	L1_IDENT	L1_IDENT
usrRate	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
negot	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
syncAsync	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
flcOnRx	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
flcOnTx	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
niClkOnRx	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
niClkOnTx	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
interRate	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
inOutBandNeg	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT

Field	ANSI 88	ANSI 92	ANSI 95
asgnrAsgne	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
logLnkNegot	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
mode	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
multiFrm	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
hdrNoHdr	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
parity	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
nmbDatBits	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
nmbStpBits	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
modemType	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
duplexMode	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
usrInfLyr2Prot	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
lyr2Ident	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
usrInfLyr3Prot	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
lyr3Ident	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
opFwdCallInd	NOT_PRESENT		
cugIntCode	NOT_PRESENT		
accTrnsprt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
businessgrp		NOT_PRESENT	NOT_PRESENT
callRef	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
cdPtyNum	**Present**	**Present**	**Present**
natAddrInd	NATNUM	NATNUM	NATNUM
numPlan	NP_ISDN	NP_ISDN	NP_ISDN
innInd	INN_ALLOW	INN_ALLOW	INN_ALLOW
addrSig	from cdPty	from cdPty	from cdPty
oddEven	from cdPtyLen	from cdPtyLen	from cdPtyLen
carrierId		NOT_PRESENT	NOT_PRESENT
carSelInf		NOT_PRESENT	NOT_PRESENT
chargeNum	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
cirAssign			NOT_PRESENT
connReq	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
egress		NOT_PRESENT	NOT_PRESENT
genAddr		NOT_PRESENT	NOT_PRESENT
genDigits		NOT_PRESENT	NOT_PRESENT
genName			NOT_PRESENT
hopCount			NOT_PRESENT
infoReqInd		NOT_PRESENT	NOT_PRESENT

Field	ANSI 88	ANSI 92	ANSI 95
jurisInf		NOT_PRESENT	NOT_PRESENT
netTransport		NOT_PRESENT	NOT_PRESENT
opServInfo			NOT_PRESENT
origCdNum		NOT_PRESENT	NOT_PRESENT
origLineInf	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
mlppPrec		NOT_PRESENT	NOT_PRESENT
redirNum	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
redirInfo	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
remotOper			NOT_PRESENT
serviceAct		NOT_PRESENT	NOT_PRESENT
serviceCode			NOT_PRESENT
specProcReq		NOT_PRESENT	NOT_PRESENT
transReq		NOT_PRESENT	NOT_PRESENT
transNetSel	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
userServInfo1			NOT_PRESENT
usr2UsrInfo	NOT_PRESENT		NOT_PRESENT

ITU Blue Book, ITU White Book, and ITU Q.767 values

Field	ITU Blue Book		ITU Q.767
natConInd	**Present**	**Present**	**Present**
satInd	SAT_NONE	SAT_NONE	SAT_NONE
contChkInd	CONTCHK_NOTREQ	CONTCHK_NOTREQ	CONTCHK_NOTREQ
echoCntrlDevInd	ECHOCDEV_NOTINCL	ECHOCDEV_NOTINCL	ECHOCDEV_NOTINCL
fwdCallInd	**Present**	**Present**	**Present**
natIntCallInd	CALL_NAT	CALL_NAT	CALL_NAT
end2EndMethInd	E2EMTH_NOMETH	E2EMTH_NOMETH	E2EMTH_NOMETH
intInd	INTIND_NOINTW	INTIND_NOINTW	INTIND_NOINTW
isdnUsrPrtInd	ISUP_USED	ISUP_USED	ISUP_USED
isdnUsrPrtPrfInd	PREF_PREFAW	PREF_PREFAW	PREF_PREFAW
isdnAccInd	ISDNACC_ISDN	ISDNACC_ISDN	ISDNACC_ISDN
sccpMethInd	SCCPMTH_NOIND	SCCPMTH_NOIND	SCCPMTH_NOIND
cdPtyNum	**Present**	**Present**	**Present**
natAddrInd	NATNUM	NATNUM	NATNUM
numPlan	NP_ISDN	NP_ISDN	NP_ISDN
innInd	INN_ALLOW	INN_ALLOW	INN_ALLOW
addrSig	from cdPty	from cdPty	from cdPty

Field	ITU Blue Book	ITU White Book	ITU Q.767
oddEven	from cdPtyLen	from cdPtyLen	from cdPtyLen
CgPtyCat	**CAT_ORD**	**CAT_ORD**	**CAT_ORD**
txMedReg	**TMR_SPEECH**	**TMR_SPEECH**	**TMR_SPEECH**
opFwdCallInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
cugIntCode	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
usrServInfo	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
accTrnsprt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
callRef	NOT_PRESENT	NOT_PRESENT	
cdPtyNum	**Present**	**Present**	**Present**
natAddrInd	NATNUM	NATNUM	NATNUM
numPlan	NP_ISDN	NP_ISDN	NP_ISDN
innInd	INN_ALLOW	INN_ALLOW	INN_ALLOW
addrSig	from cdPty	from cdPty	from cdPty
oddEven	from cdPtyLen	from cdPtyLen	from cdPtyLen
connReq	NOT_PRESENT	NOT_PRESENT	
genDigits		NOT_PRESENT	
genNmb		NOT_PRESENT	
propDly		NOT_PRESENT	
netFac		NOT_PRESENT	
notifInd		NOT_PRESENT	
orgPteCde		NOT_PRESENT	
parmCom		NOT_PRESENT	
origCdNum	NOT_PRESENT	NOT_PRESENT	
locNum		NOT_PRESENT	
mlppPrec		NOT_PRESENT	
redirgNum	NOT_PRESENT	NOT_PRESENT	
redirInfo	NOT_PRESENT	NOT_PRESENT	
remotOper		NOT_PRESENT	
serviceAct		NOT_PRESENT	
transNetSel	NOT_PRESENT	NOT_PRESENT	
txMedReqPr		NOT_PRESENT	
userServInfo		NOT_PRESENT	
usrServInfo1		NOT_PRESENT	
usr2UsrInfo	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
usr2UsrInd	NOT_PRESENT	NOT_PRESENT	

ITU97, ETSI V2, and ETSI V3 values

Field	ITU97	ETSI V2	ETSI V3
natConInd	**Present**	**Present**	**Present**
satInd	SAT_NONE	SAT_NONE	SAT_NONE
contChkInd	CONTCHK_NOTREQ	CONTCHK_NOTREQ	CONTCHK_NOTREQ
echoCntrlDevInd	ECHOCDEV_NOTINCL	ECHOCDEV_NOTINCL	ECHOCDEV_NOTINCL
fwdCallInd	**Present**	**Present**	**Present**
natIntCallInd	CALL_NAT	CALL_NAT	CALL_NAT
end2EndMethInd	E2EMTH_NOMETH	E2EMTH_NOMETH	E2EMTH_NOMETH
intInd	INTIND_NOINTW	INTIND_NOINTW	INTIND_NOINTW
isdnUsrPrtInd	ISUP_USED	ISUP_USED	ISUP_USED
isdnUsrPrtPrfInd	PREF_PREFAW	PREF_PREFAW	PREF_PREFAW
isdnAccInd	ISDNACC_ISDN	ISDNACC_ISDN	ISDNACC_ISDN
sccpMethInd	SCCPMTH_NOIND	SCCPMTH_NOIND	SCCPMTH_NOIND
cdPtyNum	**Present**	**Present**	**Present**
natAddrInd	NATNUM	NATNUM	NATNUM
numPlan	NP_ISDN	NP_ISDN	NP_ISDN
innInd	INN_ALLOW	INN_ALLOW	INN_ALLOW
addrSig	from cdPty	from cdPty	from cdPty
oddEven	from cdPtyLen	from cdPtyLen	from cdPtyLen
CgPtyCat	**CAT_ORD**	**CAT_ORD**	**CAT_ORD**
txMedReg	**TMR_SPEECH**	**TMR_SPEECH**	**TMR_SPEECH**
opFwdCallInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
cugIntCode	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
usrServInfo	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
accTrnsprt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
callRef	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
cdPtyNum	**Present**	**Present**	**Present**
natAddrInd	NATNUM	NATNUM	NATNUM
numPlan	NP_ISDN	NP_ISDN	NP_ISDN
innInd	INN_ALLOW	INN_ALLOW	INN_ALLOW
addrSig	from cdPty	from cdPty	from cdPty
oddEven	from cdPtyLen	from cdPtyLen	from cdPtyLen
connReq	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
genDigits	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
genNmb	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
propDly	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT

netFac	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
notifInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
orgPteCde	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
parmCom	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
origCdNum	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
locNum	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
mlppPrec	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
redirgNum	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
redirInfo	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
remotOper	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
serviceAct	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
transNetSel	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
txMedReqPr	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
userServInfo	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
usrServInfo1	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
usr2UsrInfo	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
usr2UsrInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
cCSS	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
netMngmtCtrls	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
cirAssMap	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
callDivTrtmnt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
cdINNum	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
callOffTrtmnt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
confTrtmnt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
uIDCapInd	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
collCallReq	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
freePhone	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
scfId	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
corrId	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT

BICC values

Field	BICC
natConInd	**Present**
satInd	SAT_NONE
contChkInd	CONTCHK_NOTEXP
echoCntrlDevInd	ECHOCDEV_NOTINCL
fwdCallInd	**Present**

Field	**BICC**
natIntCallInd	CALL_NAT
end2EndMethInd	E2EMTH_NOMETH
intInd	INTIND_NOINTW
isdnUsrPrtInd	BICC_USED
isdnUsrPrtPrfInd	BICC_PREFAW
isdnAccInd	ISDNACC_ISDN
sccpMethInd	SCCPMTH_NOIND
cdPtyNum	**Present**
natAddrInd	NATNUM
numPlan	NP_ISDN
innInd	INN_ALLOW
addrSig	from cdPty
oddEven	from cdPtyLen
CgPtyCat	**CAT_ORD**
txMedReg	**TMR_SPEECH**
opFwdCallInd	NOT_PRESENT
cugIntCode	NOT_PRESENT
usrServInfo	NOT_PRESENT
accTrnsprt	NOT_PRESENT
callRef	
cdPtyNum	**Present**
natAddrInd	NATNUM
numPlan	NP_ISDN
innInd	INN_ALLOW
addrSig	from cdPty
oddEven	from cdPtyLen
connReq	
genDigits	
genNmb	
propDly	
netFac	
notifInd	
orgPteCde	
parmCom	
origCdNum	
locNum	

Field	BICC
mlppPrec	
redirgNum	
redirInfo	
remotOper	
serviceAct	
transNetSel	
txMedReqPr	
userServInfo	
usrServInfo1	
usr2UsrInfo	NOT_PRESENT
usr2UsrInd	

NTT values

Field	NTT
natConInd	**Present**
satInd	SAT_NONE
contChkInd	CONTCHK_NOTREQ
echoCntrlDevInd	ECHOCDEV_NOTINC L
fwdCallInd	**Present**
natIntCallInd	CALL_NAT
end2EndMethInd	E2EMTH_NOMETH
intInd	INTIND_NOINTW
isdnUsrPrtInd	ISUP_USED
isdnUsrPrtPrfInd	PREF_PREFAW
isdnAccInd	ISDNACC_ISDN
sccpMethInd	SCCPMTH_NOIND
cdPtyNum	**Present**
natAddrInd	NATNUM
numPlan	NP_ISDN
innInd	INN_ALLOW
addrSig	from cdPty
oddEven	from cdPtyLen
cgPtyCat	**CAT_ORD**
txMedReg	**TMR_SPEECH**
accTrnsprt	NOT_PRESENT
cdPtyNum	**Present**
natAddrInd	NATNUM

Field	NTT
numPlan	NP_ISDN
innInd	INN_ALLOW
addrSig	from cdPty
oddEven	from cdPtyLen
genNmb	NOT_PRESENT
serviceAct	NOT_PRESENT
userServInfo	NOT_PRESENT
msgAreaInfo	NOT_PRESENT
contractorNum	NOT_PRESENT
cgNumNonNotRsn	NOT_PRESENT
addUsrId	NOT_PRESENT
carrierInfoTrans	NOT_PRESENT

Example

```
U8          *called = "8479258900";
U8          cdPty[20];
U8          cdPtyLen = 0;
U8          *calling = "8479258901";
U8          cgPty[20];
U8          cgPtyLen = 0;
S16         switchType = ST_ANSI92;
SiConEvnt   iamEvnt;

cdPtyLen = ISUPASCIItoBCD(called, cdPty, 20);
cgPtyLen = ISUPASCIItoBCD(calling, cgPty, 20);

memset(&iamEvnt,  0,  sizeof(SiConEvnt));

ISUPInitIAM( switchType, &iamEvnt, cdPty, cdPtyLen, cgPty, cgPtyLen);
```

ISUPInitINF

Initializes a SiCnStEvnt structure for transmitting an information message (INF).

Prototype

S16 TXISUPAPIFUNC **ISUPInitINF** (S16 ***switchType***, SiCnStEvnt ***event***)

Argument	Description
switchType	One of the following switch type indicators: ST_ANS88 ST_ANS92 ST_ANS95 ST_BICC ST_ITUBLUE ST_ITUWHITE
event	Pointer to the SiCnStEvnt structure to be initialized.

Details

The fields of the SiCnStEvnt structure are initialized as described in the following tables, based on the ***switchType*** parameter. Fields not described are not applicable to an INF. This function is called in preparation for a call to **ISUPConnectStatusReq**.

- ANSI 88, ANSI 92, and ANSI 95 values
- ITU Blue Book and ITU White Book values
- BICC values

ANSI 88, ANSI 92, and ANSI 95 values

Field	ANSI88	ANSI92	ANSI95
infoInd	**Present**	**Present**	**Present**
cgPtyAddrRespInd	CGPRTYADDRESP_ NOTINCL	CGPRTYADDRESP_ NOTINCL	CGPRTYADDRESP_NOTINCL
connAddrRespInd	CONNADDRNOTINCL		
redirAddrRspInd	REDIRGADDRNOTINCL		
indexRspInd	INDEXNOTINCL		
holdProvInd		CGPTYADDRSPINCLHOLD	CGPTYADDRSPINCLHOLD
cgPtyCatRespInd	CGPRTYCATRESP_ NOTINCL	CGPRTYCATRESP_ NOTINCL	CGPRTYCATRESP_NOTINCL
chrgInfoRespInd	CHRGINFO_NOTINCL	CHRGINFO_NOTINCL	CHRGINFO_NOTINCL
solInfoInd	SOLINFO_SOLICIT	SOLINFO_SOLICIT	SOLINFO_SOLICIT
mlbgInfoInd		MLBGINFONOTINCL	MLBGINFONOTINCL
accTrnspt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
businessGrp	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
callRef	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
cdPtyNum	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
cgPtyCat	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
chargeNum	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
connNum	NOT_PRESENT		
index	NOT_PRESENT		
connReq	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
origLineInf	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
redirgNum	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
redirInfo	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
usr2UsrInfo	NOT_PRESENT		NOT_PRESENT

ITU Blue Book and ITU White Book values

Field	ITU Blue Book	ITU White Book
infoInd	**Present**	**Present**
cgPtyAddrRespInd	CGPRTYADDRESP_NOTINCL	CGPRTYADDRESP_NOTINCL
holdProvInd	CGPTYADDRSP_INCLHOLD	CGPTYADDRSP_INCLHOLD
cgPtyCatRespInd	CGPRTYCATRESP_NOTINCL	CGPRTYCATRESP_NOTINCL
chrgInfoRespInd	CHRGINFO_NOTINCL	CHRGINFO_NOTINCL
solInfoInd	SOLINFO_SOLICIT	SOLINFO_SOLICIT
accTrnspt	NOT_PRESENT	
cdPtyNum	NOT_PRESENT	NOT_PRESENT
cgPtyCat	NOT_PRESENT	NOT_PRESENT
callRef	NOT_PRESENT	NOT_PRESENT
connReq	NOT_PRESENT	NOT_PRESENT
parmCom		NOT_PRESENT
netFac		NOT_PRESENT

BICC values

Field	BICC
infoInd	**Present**
cgPtyAddrRespInd	CGPRTYADDRESP_NOTINCL
holdProvInd	CGPTYADDRSP_INCLHOLD
cgPtyCatRespInd	CGPRTYCATRESP_NOTINCL
chrgInfoRespInd	CHRGINFO_NOTINCL
solInfoInd	SOLINFO_SOLICIT
accTrnspt	
cdPtyNum	NOT_PRESENT
cgPtyCat	NOT_PRESENT
callRef	NOT_PRESENT
connReq	NOT_PRESENT
parmCom	NOT_PRESENT
netFac	NOT_PRESENT

Example

```
S16        switchType=ST_ANS95;
SiAllSdus  endEvent;

ISUPInitINF(switchType, &sendEvent.m.SiCnStEvnt);
```

ISUPInitINR

Initializes a SiCnStEvnt structure for transmitting an information request message (INR).

Prototype

S16 TXISUPAPIFUNC **ISUPInitINR** (S16 ***switchType***, SiCnStEvnt ***event***)

Argument	Description
switchType	One of the following switch type indicators: ST_ANS88 ST_ANS95 ST_BICC ST_ITUBLUE ST_ITUWHITE
event	Pointer to the SiCnStEvnt structure to be initialized.

Details

The fields of the SiCnStEvnt structure are initialized as described in the following tables, based on the ***switchType*** parameter. Fields not described are not applicable to an INR. This function is called in preparation for a call to **ISUPConnectStatusReq**.

- ANSI 88, ANSI 92, and ANSI 95 values
- ITU Blue Book and ITU White Book values
- BICC values

ANSI 88, ANSI 92, and ANSI 95 values

Field	ANSI 88	ANSI 92	ANSI 95
infoReqInd	**Present**	**Present**	**Present**
cgPtyAdReqInd	CGPRTYADDREQ_ NOTREQ	CGPRTYADDREQ_ NOTREQ	CGPRTYADDREQ_ NOTREQ
connAddrRespInd	CONNADDRNOTINCL		
redirAddrRspInd	REDIRGADDRNOTINCL		
indexRspInd	INDEXNOTINCL		
holdingInd	CGPRTYADDREQ_ NOHOLD	CGPRTYADDREQ_ NOHOLD	CGPRTYADDREQ_ NOHOLD
cgPtyCatReqInd	CGPRTYCATREQ_ NOTREQ	CGPRTYCATREQ_NOTREQ	CGPRTYCATREQ_ NOTREQ
chrgInfoReqInd	CHRGINFO_NOTREQ	CHRGINFO_NOTREQ	CHRGINFO_NOTREQ
malCaIdReqInd	MALCAID_NOTREQ	MALCAID_NOTREQ	MALCAID_NOTREQ
mlbgInfoInd		MLBGINFONOTINCL	MLBGINFONOTINCL
callRef	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
connReq	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
accTrnspt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT

ITU Blue Book and ITU White Book values

Field	ITU Blue Book	ITU White Book
infoReqInd	**Present**	**Present**
cgPtyAdReqInd	CGPRTYADDREQ_NOTREQ	CGPRTYADDREQ_NOTREQ
holdingInd	CGPRTYADDREQ_NOHOLD	CGPRTYADDREQ_NOHOLD
cgPtyCatReqInd	CGPRTYCATREQ_NOTREQ	CGPRTYCATREQ_NOTREQ
chrgInfoReqInd	CHRGINFO_NOTREQ	CHRGINFO_NOTREQ
malCaIdReqInd	MALCAID_NOTREQ	MALCAID_NOTREQ
callRef	NOT_PRESENT	NOT_PRESENT
parmCom		NOT_PRESENT
netFac		NOT_PRESENT

BICC values

Field	BICC
infoReqInd	**Present**
cgPtyAdReqInd	CGPRTYADDREQ_NOTREQ
holdingInd	CGPRTYADDREQ_NOHOLD
cgPtyCatReqInd	CGPRTYCATREQ_NOTREQ
chrgInfoReqInd	CHRGINFO_NOTREQ
malCaIdReqInd	MALCAID_NOTREQ
callRef	NOT_PRESENT
parmCom	NOT_PRESENT
netFac	NOT_PRESENT

Example

```
S16        switchType=ST_ANS95;
SiAllSdus  sendEvent;

ISUPInitINR(switchType, &sendEvent.m.SiCnStEvnt);
```

ISUPInitREL

Initializes a SiRelEvnt structure for transmitting a release message (REL).

Prototype

S16 TXISUPAPIFUNC **ISUPInitREL** (S16 ***switchType***, SiRelEvnt ***event**, U8 ***cause***)

Argument	Description
switchType	One of the following switch type indicators: ST_ANS88 ST_ANS92 ST_ANS95 ST_BICC ST_ITUBLUE ST_JNTT ST_Q767
event	Pointer to the SiRelEvnt structure to be initialized.
cause	Cause value.

Details

The fields of the SiRelEvnt structure are initialized as described in the following tables, based on the ***switchType*** parameter. Fields not described are not applicable to an REL. This function is called in preparation for a call to **ISUPReleaseReq**.

- ANSI 88, ANSI 92, and ANSI 95 values
- ITU Blue Book, ITU White Book, and ITU Q.767 values
- BICC values
- NTT values

ANSI 88, ANSI 92, and ANSI 95 values

Field	ANSI 88	ANSI 92	ANSI 95
causeDgn	**Present**	**Present**	**Present**
location	ILOC_PRIVNETLU	ILOC_PRIVNETLU	ILOC_PRIVNETLU
cdeStand	CSTD_NAT	CSTD_NAT	CSTD_NAT
causeVal	from cause	from cause	from cause
dgnVal	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
accTrnspt	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
autoCongLvl		NOT_PRESENT	NOT_PRESENT
callRef	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
chargeNum		NOT_PRESENT	NOT_PRESENT
cugIntCode	NOT_PRESENT		
genAddr		NOT_PRESENT	NOT_PRESENT
redirInfo	NOT_PRESENT		
redirNum	NOT_PRESENT		
redirgNum	NOT_PRESENT		
serviceAct			NOT_PRESENT
sigPointCode	NOT_PRESENT		
usr2UsrInfo	NOT_PRESENT		NOT_PRESENT

ITU Blue Book, ITU White Book, and ITU Q.767 values

Field	ITU Blue Book	ITU White Book	ITU Q.767
causeDgn	**Present**	**Present**	**Present**
location	ILOC_PRIVNETLU	ILOC_PRIVNETLU	ILOC_PRIVNETLU
cdeStand	CSTD_NAT	CSTD_NAT	CSTD_NAT
causeVal	from cause	from cause	from cause
dgnVal	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
redirInfo	NOT_PRESENT	NOT_PRESENT	
redirNum	NOT_PRESENT	NOT_PRESENT	
accTrnspt	NOT_PRESENT	NOT_PRESENT	
sigPointCode	NOT_PRESENT	NOT_PRESENT	
usr2UsrInfo	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
autoCongLvl	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT
netFac		NOT_PRESENT	
accDelInfo		NOT_PRESENT	
parmCom		NOT_PRESENT	
redirRstr		NOT_PRESENT	
usr2UsrInd		NOT_PRESENT	

BICC values

Field	BICC
causeDgn	**Present**
location	ILOC_PRIVNETLU
cdeStand	CSTD_NAT
causeVal	from cause
dgnVal	NOT_PRESENT
redirInfo	NOT_PRESENT
redirNum	NOT_PRESENT
accTrnspt	NOT_PRESENT
sigPointCode	NOT_PRESENT
usr2UsrInfo	NOT_PRESENT
autoCongLvl	NOT_PRESENT
netFac	NOT_PRESENT
accDelInfo	NOT_PRESENT
parmCom	NOT_PRESENT
redirRstr	NOT_PRESENT
usr2UsrInd	NOT_PRESENT

NTT values

Field	NTT
causeDgn	**Present**
location	ILOC_PRIVNETLU
cdeStand	CSTD_NAT
causeVal	from cause
dgnVal	NOT_PRESENT
usr2UsrInfo	NOT_PRESENT
ServiceAct	NOT_PRENENT
CgPtyNum	

Example

```
S16         switchType=ST_ITUWHITE;
SiAllSdus   sendEvent;
U8          cause= CCCALLCLR;    /* Normal call clearing, defined in iedefs.h */;

ISUPInitREL(switchType, &sendEvent.m.SiRelEvnt, cause);
```

ISUPInitRES

Initializes a SiResmEvnt structure for transmitting a resume message (RES).

Prototype

S16 TXISUPAPIFUNC **ISUPInitRES** (S16 ***switchType***, SiResmEvnt ***event**)

Argument	Description
switchType	One of the following switch type indicators: ST_ANS88 ST_ANS92 ST_ANS95 ST_BICC ST_ITUBLUE ST_ITUWHITE ST_Q767
event	Pointer to the SiResmEvnt structure to be initialized.

Details

The fields of the SiResmEvnt structure are initialized as described in the following tables, based on the ***switchType*** parameter. Fields not described are not applicable to an RES. This function is called in preparation for a call to **ISUPResumeReq**.

- ANSI 88, ANSI 92, and ANSI 95 values
- ITU Blue Book, ITU White Book, and ITU Q.767 values
- BICC values

ANSI 88, ANSI 92, and ANSI 95 values

Field	ANSI 88	ANSI 92	ANSI 95
susResInd	**Present**		
susResInd	SR_ISDNSUBINIT	SR_ISDNSUBINIT	SR_ISDNSUBINIT
callRef	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT

ITU Blue Book, ITU White Book, and ITU Q.767 values

Field	ITU Blue Book	ITU White Book	ITU Q.767
susResInd	**Present**		
susResInd	SR_ISDNSUBINIT	SR_ISDNSUBINIT	SR_ISDNSUBINIT
callRef	NOT_PRESENT	NOT_PRESENT	

BICC values

Field	BICC
susResInd	
susResInd	SR_ISDNSUBINIT
callRef	NOT_PRESENT

Example

```
S16        switchType=ST_ITUWHITE;
SiAllSdus  sendEvent;

ISUPInitRES(switchType, &sendEvent.m.SiResmEvnt);
```

ISUPInitSAM

Initializes a SiCnStEvnt structure for transmitting a subsequent address message (SAM).

Prototype

S16 TXISUPAPIFUNC **ISUPInitSAM** (S16 ***switchType***, SiCnStEvnt ***event***, U8 ***subAddr***, U8 ***subAddrLen***)

Argument	Description
switchType	ST_BICC ST_ITUBLUE ST_Q767
event	Pointer to the SiCnStEvnt structure to be initialized.
subAddr	Pointer to BCD subsequent address.
subAddrLen	Number of BCD digits in subsequent address.

Details

The fields of the SiCnStEvnt structure are initialized as described in the following table, based on the ***switchType*** parameter. Fields not described are not applicable to an SAM. This function is called in preparation for a call to **ISUPConnectStatusReq**.

- ITU Blue Book, ITU White Book, and ITU Q.767 values
- BICC values

ITU Blue Book, ITU White Book, and ITU Q.767 values

Field	ITU Blue Book	ITU White Book	ITU Q.767
subNum	**Present**	**Present**	**Present**
oddEven	from subAddrLen	from subAddrLen	from subAddrLen
addrSig	from subAddr	from subAddr	from subAddr

BICC values

Field	BICC
subNum	**Present**
oddEven	from subAddrLen
addrSig	from subAddr

Example

```
U8          _sAddr[20];
U8          _sAddrLen = 0;
S16         switchType = ST_ANSI92;
SiCnStEvnt  samEvnt;

memset(&samEvnt,  0,  sizeof(SiCnStEvnt));


_sAddrLen = ISUPASCIItoBCD( _addr, _sAddr, 20 );
ISUPInitSAM( switchType, &samEvnt, _sAddr, _sAddrLen );
```

ISUPInitSUS

Initializes a SiSuspEvnt structure for transmitting a suspend message (SUS).

Prototype

S16 TXISUPAPIFUNC **ISUPInitSUS** (S16 ***switchType***, SiSuspEvnt ***event**)

Argument	Description
switchType	One of the following switch type indicators: ST_ANS88 ST_ANS92 ST_BICC ST_ITUBLUE ST_Q767
event	Pointer to the SiSuspEvnt structure to be initialized.

Details

The fields of the SiSuspEvnt structure are initialized as described in the following tables, based on the ***switchType*** parameter. Fields not described are not applicable to an SUS. This function is called in preparation for a call to **ISUPSuspendReq**.

- ANSI 88, ANSI 92, and ANSI 95 values
- ITU Blue Book, ITU White Book, and ITU Q.767 values
- BICC values

ANSI 88, ANSI 92, and ANSI 95 values

Field	ANSI 88	ANSI 92	ANSI 95
susResInd	**Present**	**Present**	**Present**
susResInd	SR_ISDNSUBINIT	SR_ISDNSUBINIT	SR_ISDNSUBINIT
callRef	NOT_PRESENT	NOT_PRESENT	NOT_PRESENT

ITU Blue Book, ITU White Book, and ITU Q.767 values

Field	ITU Blue Book	ITU White Book	ITU Q.767
susResInd	**Present**	**Present**	**Present**
susResInd	SR_ISDNSUBINIT	SR_ISDNSUBINIT	SR_ISDNSUBINIT
callRef	NOT_PRESENT	NOT_PRESENT	

BICC values

Field	BICC
susResInd	**Present**
susResInd	SR_ISDNSUBINIT
callRef	NOT_PRESENT

Example

```
S16         switchType=ST_ITUWHITE;
SiAllSdus   sendEvent;

ISUPInitSUS(switchType, &sendEvent.m.SiSuspEvnt);
```

7 ISUP management function reference

ISUP management function summary

NaturalAccess™ ISUP provides the following types of management functions:

- Configuration
- Control
- Statistics and Status

All of these functions are synchronous functions, so they block the calling application while waiting for a response.

Configuration functions

Function	Description
isupCircCfg	
isupGenCfg	Sends a general configuration buffer to the TX board.
isupGetCircCfg	Sends a get circuit configuration request to the TX board.
isupGetGenCfg	
isupGetNSapCfg	Sends a get network service access point configuration request to the TX board.
isupGetUSapCfg	Sends a get configuration request to the TX board.
isupInitCircCfg	Builds a circuit configuration buffer that can be passed to **isupCircCfg**.
	Builds a general configuration buffer that can be passed to **isupGenCfg**.
	Builds a network service access point configuration buffer that can be passed to **isupNSapCfg**.
isupInitUSapCfg	Builds a user service access point configuration buffer that can be passed to **isupUSapCfg**.
isupNSapCfg	Sends a network service access point configuration buffer to the TX board.
isupUSapCfg	Sends a user service access point configuration buffer to the TX board.

Control functions

Function	Description
	Sends a request to block the given circuit.
isupDeleteCircuit	Sends a request to delete the given circuit.
isupInitMgmtAPI	Initializes ISUP management and opens a channel to the TX board.
isupResetCircuit	Sends a request to reset the given circuit.
isupTermMgmtAPI	Closes ISUP management and the channel to the TX board.
isupTraceControl	Sends a request to control what is saved to the trace log.
isupUnblockCircuit	Sends a request to unblock the given circuit.
isupValidateCircuit	Sends a request to validate the given circuit .

Statistics and status functions

Function	Description
isupCircuitStats	Sends a request to retrieve the statistics for a given circuit.
isupCircuitStatsEx	Sends a request to retrieve the statistics for a given circuit and lets you control whether the statistics counters are reset.
isupCircuitStatus	Sends a request to retrieve the status for a given circuit.
isupNSapStats	Sends a request to retrieve the statistics for a given network service access point.
isupNSapStatsEx	Sends a request to retrieve the statistics for a given network service access point and gives you control whether the statistics counters are reset.

Using the ISUP management function reference

This section provides an alphabetical reference to the ISUP management functions. A typical function includes:

Prototype	The prototype is followed by a list of the function arguments. Data types include: • U8 (8-bit unsigned) • S16 (16-bit signed) • U16 (16-bit unsigned) • U32 (32-bit signed) • U32 (32-bit unsigned) • Bool (8-bit unsigned) If a function argument is a data structure, the complete data structure is defined.
Return values	The return value for a function is either ISUP_SUCCESS or an error code.

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

isupBlockCircuit

Sends a request to block the given circuit and blocks the calling application while waiting for a response.

Note: This function is not supported in BICC. The ISUP stack layer reports a service unavailable alarm message if this function is invoked for BICC.

Prototype

short **isupBlockCircuit** (U8 ***board***, U32 ***circId***)

Argument	Description
board	TX board number.
	Circuit ID to block.

Return values

Return value	Description
ISUP_SUCCESS	
ISUP_BOARD	***board*** is out of range.
ISUP_DRIVER	Error occurred accessing the driver.
	Task on the TX board returned a failure.
ISUP_UNBOUND	

Example

```
S16  status;
U8   boardNum =1;
U32  circuitId=1;

if ((status = isupBlockCircuit(boardNum, circuitId)) != ISUP_MGMT_SUCCESS)
{
    printf( "Board %d Circuit %d Block Request failed: status = %d\n", boardNum,
        circuitId, status );
}
else
{
    printf("Successfully blocked circuit %d on board %d\n", circuitId, boardNum );
}
```

isupCircCfg

Sends a circuit configuration buffer to the TX board and blocks the calling application while waiting for a response.

Prototype

short **isupCircCfg** (U8 ***board***, IsupCircCfg ****cfg***)

Argument	Description
	TX board number.
cfg	structure definition.

Return values

Return value	Description
ISUP_BOARD	
ISUP_DRIVER	Error occurred accessing the driver.
ISUP_FAILED	Task on the TX board returned a failure.
ISUP_UNBOUND	Application failed to call **isupInitMgmtAPI** prior to this call.

Details

An application must set the field values in the IsupCircCfg structure before calling **isupCircCfg**. Set the values in any of the following ways:

- Call **isupInitCircCfg** to set the fields to default values.
- Set each field value from within the application.
- Call **isupInitCircCfg** and then override specific field values before passing the IsupCircCfg structure to **isupCircCfg**.

isupCircCfg is typically called once for each configured circuit.

Example

```
S16            status;
U8             boardNum = 1;
IsupCircCfg    cfg;

/* Populate IsupCircCfg structure before calling isupCircCfg */

if ((status = isupCircCfg(boardNum, &cfg)) != ISUP_MGMT_SUCCESS)
{
    printf( "Board %d Semd Circuit Configuration Request failed: status = %d\n",
        boardNum, status );
}
else
    printf( "Successfully sent board circuit information to board %d\n", boardNum );
}
```

isupCircuitStats

Sends a request to retrieve the statistics for a given circuit and blocks the calling application while waiting for a response.

Prototype

short **isupCircuitStats** (U8 ***board***, U32 ***circId***, IsupCircStats ****stats***)

Argument	Description
board	TX board number.
circId	Circuit for which to retrieve statistics.
stats	Pointer to the following IsupCircStats structure, where the requested statistics information is returned:

```
typedef struct _IsupCircStats
{
  DateTime dt;             /* Date and time                              */
  Duration dura;           /* Duration                                   */
  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  cirGrBlockTx;       /* Circuit Group Blocking Transmitted         */
  S32  cirGrBlockAckTx;    /* Circuit Group Blocking Ack Transmitted     */
  S32  cirGrUnBlockTx;     /* Circuit Group Unblocking Transmitted       */
  S32  cirGrUnBlockAckTx;  /* Circuit Group Unblocking Ack Transmitted   */
  S32  cirGrQTx;           /* Circuit Group Query Transmitted            */
  S32  cirGrQAckTx;        /* Circuit Group Query Acknowledge Transmitted*/
  S32  cirGrResTx;         /* Circuit Group Reset Transmitted            */
  S32  cirGrResAckTx;      /* Circuit Group Reset Ack Transmitted        */
  S32  usrPrtTstTx;        /* User Part Test Transmitted                 */
  S32  usrPrtAvTx;         /* User Part Available Transmitted            */
  S32  cirValTstTx;        /* Circuit Validation Test Transmitted        */
  S32  cirValRspTx;        /* Circuit Validation Response 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  cirGrBlockRx;       /* Circuit Group Blocking Received            */
  S32  cirGrBlockAckRx;    /* Circuit Group Blocking Ack Received        */
  S32  cirGrUnBlockRx;     /* Circuit Group Unblocking Received          */
  S32  cirGrUnBlockAckRx;  /* Circuit Group Unblocking Ack Received      */
  S32  cirGrQRx;           /* Circuit Group Query Received               */
  S32  cirGrQAckRx;        /* Circuit Group Query Acknowledge Transmitted*/
  S32  cirGrResRx;         /* Circuit Group Reset Received               */
  S32  cirGrResAckRx;      /* Circuit Group Reset Ack Received           */
  S32  usrPrtTstRx;        /* User Part Test Received                    */
  S32  usrPrtAvRx;         /* User Part Available Received               */
  S32  cirValTstRx;        /* Circuit Validation Test received           */
  S32  cirValRspRx;        /* Circuit Validation Response received       */
} IsupCircStats ;
```

Return values

Return value	Description
ISUP_SUCCESS	
ISUP_BOARD	***board*** is out of range.
ISUP_DRIVER	Error occurred accessing the driver.
ISUP_FAILED	Task on the TX board returned a failure.
ISUP_UNBOUND	Application failed to call **isupInitMgmtAPI** prior to this call.

Details

isupCircuitStats automatically resets the statistics counters. To control the resetting of the statistics counters, use **isupCircuitStatsEx** instead of **isupCircuitStats**. For information, see *isupCircuitStatsEx* on page 138**.**

Example

```
S16             status;
U8              boardNum = 1;
U32             circId = 1;
IsupCircStats   sts;

if ((status = isupCircuitStats(boardNum, circId, &sts)) != ISUP_MGMT_SUCCESS)
{
    printf( "Board %d Circuit %d Statistics Request failed: status = %d\n",
        boardNum, circId, status );
}
else
    printf( "Successfully retrieved circuit statistics for circuit %d on board %d\n",
        circId, boardNum );

/* The sts structure contains the circuit statistics */

}
```

isupCircuitStatsEx

Sends a request to retrieve the statistics for a given circuit and blocks the calling application while waiting for a response. This function also controls whether the statistics counters are reset after the function execution.

Prototype

short **isupCircuitStatsEx** (U8 ***board***, U32 ***circId***, IsupCircStats ****stats***, U8 ***reset***)

Argument	Description
board	TX board number.
circId	Circuit for which to retrieve statistics.
stats	Pointer to the IsupCircStats structure where the requested statistics information is returned. Refer to *isupCircuitStats* on page 136 for the structure definition.
reset	Indicates whether to reset the statistics counters after function execution: 0 = Reset the statistics counters to zeroes. 1 = Do not reset the statistics counters.

Return values

Return value	Description
ISUP_SUCCESS	
ISUP_BOARD	***board*** is out of range.
ISUP_DRIVER	Error occurred accessing the driver.
ISUP_FAILED	Task on the TX board returned a failure.
	Application failed to call **isupInitMgmtAPI** prior to this call.

Examples

```
S16            status;
U8             boardNum = 1;
U32            circId = 1;
IsupCircStats  sts;
U8             resetStats=1;

if ((status = isupCircuitStatsEx(boardNum, circId, &sts, resetStats)) !=
        ISUP_MGMT_SUCCESS)
{
    printf( "Board %d Circuit %d Statistics Request failed: status = %d\n",
        boardNum, circId, status );
}
else
    printf( "Successfully retrieved circuit statistics for circuit %d on board %d\n",
        circId, boardNum );

/* The sts structure contains the circuit statistics information
}
```

isupCircuitStatus

Sends a request to retrieve the status for a given circuit and blocks the calling application while waiting for a response.

Prototype

short **isupCircuitStatus** (U8 ***board***, U32 ***circId***, IsupCircStatus ***status***)

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

status	Pointer to the following IsupCircStatus structure where the requested status information is returned:
	`typedef struct _IsupCircStatus        /* ISUP Circuit status   */`
	`{`
	`  DateTime dt;                /* Date and time              */`
	`  U8 transState;              /* Circuit transient state    */`
	`  U8 callState;               /* Circuit call processing state */`
	`  U16 fill1;`
	`} IsupCircStatus;`

Return values

Return value	Description
ISUP_SUCCESS	
ISUP_BOARD	***board*** is out of range.
ISUP_DRIVER	Error occurred accessing the driver.
ISUP_FAILED	Task on the TX board returned a failure.
ISUP_UNBOUND	Application failed to call **isupInitMgmtAPI** prior to this call.

Example

```
S16             status;
U8              boardNum = 1;
U32             circId = 1;
IsupCircStatus  circSta;

if ((status = isupCircuitStatus(boardNum, circuitId, &circSta)) != ISUP_MGMT_SUCCESS)
    printf( "Board %d Circuit %d Status Request failed: status = %d\n",
        boardNum, circuitId, status );
else
    printf( "Successfully retrieved circuit status for circuit %d on board %d\n",
        circuitId, boardNum );

/* The circSta structure contains the circuit status information */

}
```

isupDeleteCircuit

Sends a request to delete the given circuit and blocks the calling application while waiting for a response.

Prototype

short **isupDeleteCircuit** (U8 ***board***, U32 ***circId***)

Argument	Description
board	TX board number.
circId	Identifier of the circuit to delete.

Return values

Return value	Description
ISUP_SUCCESS	
ISUP_BOARD	***board*** is out of range.
ISUP_DRIVER	Error occurred accessing the driver.
ISUP_FAILED	Task on the TX board returned a failure.
ISUP_UNBOUND	Application failed to call **isupInitMgmtAPI** prior to this call.

Example

```
S16  status;
U8   boardNum=1;
U32  circuitId=1;

if ((status = isupDeleteCircuit(boardNum, circuitId)) != ISUP_MGMT_SUCCESS)
{
    printf( "Board %d Circuit %d Delete Request failed: status = %d\n", boardNum,
        circuitId, status );
}
else
{
    printf("Successfully deleted circuit %d on board %d\n", circuitId
        boardNum );
}
```

isupGenCfg

Sends a general configuration buffer to the TX board and blocks the calling application while waiting for a response.

Prototype

short **isupGenCfg** (U8 ***board***, IsupGenParms ***cfg***)

Argument	Description
board	
cfg	Pointer to the IsupGenParms structure to send. Refer to *isupInitGenCfg* on page 148 for the structure definition.

Return values

Return value	Description
ISUP_SUCCESS	
ISUP_BOARD	***board*** is out of range.
ISUP_DRIVER	Error occurred accessing the driver.
ISUP_FAILED	Task on the TX board returned a failure.
	Application failed to call **isupInitMgmtAPI** prior to this call.

Details

An application must set the field values in the IsupGenParms structure before calling **isupGenCfg**. Set the values in any of the following ways:

- Call **isupInitGenCfg** to set the fields to default values.
- Set each field value from within the application.
- Call **isupInitGenCfg** and then override specific field values before passing the IsupGenParms structure to **isupGenCfg**.

Example

```
S16           status;
U8            boardNum = 1;
IsupGenParms  cfg;

/* Populate cfg structure before calling isupGenCfg function */

if ((status = isupGenCfg(boardNum, &cfg)) != ISUP_MGMT_SUCCESS)
{
    printf( "Board %d Send General Configuration Request failed: status = %d\n",
        boardNum, status );
}
else
    printf( "Successfully sent general configuration information to board %d\n",
        boardNum );
}
```

isupGetCircCfg

Sends a get configuration request to the TX board and blocks the calling application while waiting for a response. A board circuit configuration is returned. The received configuration parameters are placed in the configuration buffer supplied by the application.

Prototype

short **isupGetCircCfg** (U8 ***board***, IsupCircCfg ****cfg***, U32 ***circuitId***)

Argument	Description
board	TX board number.
	Pointer to the IsupCircCfg structure where the requested configuration information is returned. Refer to *isupInitCircCfg* on page 146 for the structure definition.
circuitId	Circuit ID number to use.

Return values

Return value	Description
ISUP_BOARD	
ISUP_DRIVER	Error occurred accessing the driver.
ISUP_FAILED	Task on the TX board returned a failure.
ISUP_UNBOUND	Application failed to call **isupInitMgmtAPI** prior to this call.

Example

```
S16          status;
8            boardNum=1;
IsupCircCfg  cfg;
U32          circuitId=1;

if ((status = isupGetCircCfg(boardNum, &cfg, circuitId)) != ISUP_MGMT_SUCCESS)
{
    printf( "Board %d Circuit %d Get Configuration Request failed: status = %d\n",
        boardNum, circuitId, status );
}
else
{
    printf("Successfully obtained configuration information for circuit %d on
        board %d\n", circuitId, boardNum);

  /* The cfg structure contains the circuit configuration information */

}
```

isupGetGenCfg

Sends a get configuration request to the TX board and blocks the calling application while waiting for a response. A general configuration for the board is returned. The received configuration parameters are placed in the configuration buffer supplied by the application.

Prototype

short **isupGetGenCfg** (U8 ***board***, IsupGenParms ****cfg***)

Argument	Description
board	TX board number.
cfg	Pointer to the IsupGenParms structure where the requested configuration information is returned. Refer to *isupInitGenCfg* on page 148 for the structure definition.

Return values

Return value	Description
ISUP_SUCCESS	
ISUP_BOARD	***board*** is out of range.
ISUP_DRIVER	Error occurred accessing the driver.
ISUP_FAILED	Task on the TX board returned a failure.
ISUP_UNBOUND	Application failed to call **isupInitMgmtAPI** prior to this call.

Example

```
S16           status;
U8            boardNum = 1;
IsupGenParms  cfg;

if ((status = isupGetGenCfg(boardNum, &cfg)) != ISUP_MGMT_SUCCESS)
{
    printf( "Board %d Send General Configuration Request failed: status = %d\n",
        boardNum, status );
}
else
    printf( "Successfully sent general configuration information to board %d\n",
        boardNum );
/* The cfg structure contains the returned general configuration information */

}
```

isupGetNSapCfg

Sends a get configuration request to the TX board and blocks the calling application while waiting for a response. A network service access point configuration for the board is returned. The received configuration parameters are placed in the configuration buffer supplied by the application.

Prototype

short **isupGetNSapCfg** (U8 ***board***, IsupNSapCfg ***cfg***, S16 ***sapId***)

Argument	Description
board	TX board number.
cfg	Pointer to the IsupNSapCfg structure where the requested configuration information is returned. Refer to *isupInitNSapCfg* on page 154 for the structure definition.
sapId	ISUP service access point.

Return values

Return value	Description
ISUP_SUCCESS	
ISUP_BOARD	***board*** is out of range.
ISUP_DRIVER	Error occurred accessing the driver.
ISUP_FAILED	Task on the TX board returned a failure.
ISUP_UNBOUND	Application failed to call **isupInitMgmtAPI** prior to this call.

Example

```
S16         status;
U8          boardNum=1;
IsupNsapCfg cfg;
S16         sapId=1;


if ((status = isupGetNsapCfg(boardNum, &cfg, sapId)) != ISUP_MGMT_SUCCESS)
{
    printf( "Board %d Service Access Point %d Get NSAP Configuration Request
        failed: status = %d\n", boardNum, sapId, status );
}
else
{
    printf("Successfully obtained NSAP configuration information for service access
        point %d on board %d\n", sapId, boardNum);

 /* The cfg structure contains the returned NSAP configuration information */

}
```

isupGetUSapCfg

Sends a get configuration request to the TX board and blocks the calling application while waiting for a response. A user service access point configuration for the board is returned. The received configuration parameters are placed in the configuration buffer supplied by the application.

Prototype

short **isupGetUSapCfg** (U8 ***board***, IsupUSapCfg ****cfg***, U16 ***sapId***)

Argument	Description
board	TX board number.
cfg	Pointer to the IsupUSapCfg structure where the requested configuration information is returned. Refer to *isupInitUSapCfg* on page 155 for the structure definition.
sapId	

Return values

Return value	Description
ISUP_BOARD	***board*** is out of range.
ISUP_DRIVER	Error occurred accessing the driver.
ISUP_FAILED	Task on the TX board returned a failure.
ISUP_UNBOUND	Application failed to call **isupInitMgmtAPI** prior to this call.

Example

```
S16          status;
U8           boardNum=1;
IsupUsapCfg  cfg;
S16          sapId=1;


if ((status = isupGetUsapCfg(boardNum, &cfg, sapId)) != ISUP_MGMT_SUCCESS)
{
    printf( "Board %d Service Access Point %d Get USAP Configuration Request
        failed: status = %d\n", boardNum, sapId, status );
}
else
{
    printf("Successfully obtained USAP configuration information for service access
        point %d on board %d\n", sapId, boardNum);

/* The structure cfg structure contains the returned USAP configuration information */

}
```

isupInitCircCfg

Builds a circuit configuration buffer that can be passed to **isupCircCfg**.

Prototype

void **isupInitCircCfg** (IsupCircCfg ***cfg**, U32 ***dpc***)

Argument	Description

cfg	Pointer to the following IsupCircCfg structure to initialize:
	`typedef struct _IsupCircCfg`
	`{`
	`U32 circuitId; /* Circuit id code */`
	`U16 cic; /* cic */`
	`U16 switchType; /* Switch type of this circuit */`
	`U32 dstPointCode; /* Destination point code */`
	`U32 altOrgPointCode /* Origination point code */`
	`U8 controlType; /* Type of control */`
	`U8 circuitType; /* Type of circuit */`
	`U8 bearerProfile; /* Bearer profile */`
	`U8 groupChars; /* Alignment */`
	`Bool contCheckReq; /* Continuity check required */`
	`Bool nonSS7Con; /* Connecting to non SS7 network */`
	`U8 ssf; /* SSF, if this is non - 0xff, then use */`
	`/* it instead */`
	`U8 fill2; /* Alignment */`
	`IsupAddr outTrunkGrp; /* Outgoing trunk group number */`
	`IsupAddr comLLID; /* Common language location identifier */`
	`TimerCfg t4; /* t4 timer - user part test sent */`
	`TimerCfg t12; /* t12 timer - blocking sent */`
	`TimerCfg t13; /* t13 timer - initial blocking sent */`
	`TimerCfg t14; /* t14 timer - unblocking sent */`
	`TimerCfg t15; /* t15 timer - initial unblocking sent */`
	`TimerCfg t37; /* t37 timer - waiting for receipt of */`
	`/* message after user part available */`
	`/* test started */`
	`TimerCfg tPause; /* tPause timer - MTP Pause received */`
	`TimerCfg tVal; /* Circuit validation timer */`
	`} IsupCircCfg;`
	Refer to the Details section for more information.

dpc	Destination point code.

Details

After calling **isupInitCircCfg**, call **isupCircCfg** to set the circuit configuration. You can optionally override specific field values before calling **isupCircCfg**.

The fields in the IsupCircCfg structure are initialized with the following values:

Field	Description
circuitId	Number of the circuit. This value must be unique for all defined circuits and be less than the value used in the ***maxCircuits*** field of **isupGenCfg**. The application and the ISUP layer use this number to identify circuits, but it has no meaning to the far exchange. Default = 1.
cic	Circuit identification code (CIC). This number must agree with the CIC assigned to this circuit at the far exchange. Default = 1.
dstPointCode	Destination point code to where this circuit connects, initialized from the ***dpc*** argument.

Field	Description
altOrgPointCode	Alternate originating point code for the circuit. Use this parameter when configuring the board to act as multiple originating point codes (OPCs). The OPC must be properly configured in MTP for the new ISUP OPC to work. For more information on configuring multiple OPCs, see the *Dialogic® NaturalAccess™ Signaling Software Configuration Manual*. Default = 0 (configuration uses the ISUP general configuration originating point code).
controlType	One of the following circuit control values: ISUP_CTL_NONE ISUP_CTL_ALL ISUP_CTL_ODDEVEN
circuitType	One of the following circuit group usage values: ISUP_CIR_INCOMING ISUP_CIR_OUTGOING ISUP_CIR_BOTHWAY Default = ISUP_CIR_INCOMING.
	Time to wait between user part test messages. value = 0 enable = FALSE
t12	Time to wait for response for a transmitted blocking message. value = 12 enable = TRUE
t13	Time to wait for a response for the initially transmitted blocking message. value = 60 enable = TRUE
t14	Time to wait for a response for a transmitted unblocking message. value = 12 enable = TRUE
t15	Time to wait for a response for the initially transmitted unblocking message. value = 60 enable = TRUE
	ANSI circuit validation timer. value = 30 enable = TRUE

Example

```
IsupCircCfg  cfg;
U32  dpc = 1.1.2;

isupInitCircCfg(&cfg, dpc);
```

isupInitGenCfg

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

Prototype

void **isupInitGenCfg** (IsupGenParms ***cfg**, U32 ***opc***)

Argument	Description
cfg	Pointer to the IsupGenParms structure to initialize. For more information, refer to *IsupGenParms structure* on page 149 and to the Details that follow.
opc	Originating point code.

IsupGenParms structure

```
typedef struct _IsupGenParms
{
  U16       maxSaps;       /* Max number of ISUP user saps         */
  U16       maxNetSaps;    /* Max number of network saps (MTP3)    */
  U16       maxCircGrp;    /* Max number of circuit groups         */
  U16       maxRoutes;     /* Max number of routes                 */
  U32       maxCircuits;   /* Max number of circuits               */
  U32       maxCallRefs;   /* Max number of call references        */
  U32       orgPointCode;
  Bool      sccpSupport;
  U8        ituUCICs;
  S16       poolUpper;
  S16       poolLower;
  S16       timerRes;      /* Timer resolution                     */
  U16       qCongOnset1;
  U16       qCongAbate1;
  U16       qCongOnset2;
  U16       qCongAbate2;
  U16       qCongOnset3;
  U16       qCongAbate3;
  U16       mCongOnset1;
  U16       mCongAbate1;
  U16       mCongOnset2;
  U16       mCongAbate2;
  U16       mCongOnset3;
  U16       mCongAbate3;
  IsupAddr  comLLID;       /* Common language location identifier  */
  TimerCfg  t18;
  TimerCfg  t19;
  TimerCfg  t20;
  TimerCfg  t21;
  TimerCfg  t22;
  TimerCfg  t23;
  TimerCfg  t28;
  TimerCfg  tFrstGrpRx;    /* First group received timer            */
  TimerCfg  tGrpReset;     /* Group reset timer                     */
  PDesc     stkMgr;
  U32   traceFlags;
  U32   txmonTimeout;   /* Sent to TXMON during PAUSE, RESUME,      */
                        /*   and other large operations to ask for  */
                        /*   more time to report                    */
  U8    igPassAlng;     /* Don't use Pass Along messaging with end  */
                        /* to end connection                        */
  U8    extElmts;       /* Send up unrecognized I.E.s as extended   */
                        /*   I.E.s                                  */
  U8    rawMsgs;        /* Send up unrecognized messages as raw     */
                        /* message indications                      */
  U8    oneGrpMsg;      /* If true, should react to first group     */
                        /*   message                                */
  U8    rmtUsrUnavl;    /* If true, should start appropriate User   */
                        /*   Part Test procedure                    */
  U8    grpResetEvent;  /* Send up one group reset event instead of */
                        /*  many separate circuit reset events      */
  U8    slsFromCICs;    /* Set the ANSI SLS value to the bottom     */
                        /*  bots of the CIC (default TRUE)          */
  U8    spare1;         /* spare                                    */
  U8    dsblRmtUsrUnavl;/* If true, should disable appropriate      */
                        /* User Part Test procedure might be SSURN  */
                        /*   specific                               */
  U8    restartT7;      /* Restart T7 when an ibound INR is         */
                        /  received                                 */
                        /* Might beSSURN specific                   */
  U8    disableACL;     /* Disable automatic congestion control     */
  U8    spare2;         /* Spare                                    */
} IsupGenParms;
```

Details

After calling **isupInitGenCfg**, call **isupGenCfg** to set the general ISUP configuration. You can optionally override specific field values before calling **isupGenCfg**.

The fields of the IsupGenParms structure are initialized with the following values:

Field	Description
maxSaps	Maximum number of applications. Default = 2.
maxNetSaps	Maximum number of interfaces with the MTP3 network layer. Default = 2.
maxCircuits	Maximum number of circuits managed by the ISUP layer. Default = 96.
maxCircGrp	Maximum number of simultaneous group requests managed by the ISUP layer. Default = 32.
maxCallRefs	Maximum number of call references, and hence connections, that ISUP can keep track of simultaneously. Default = 16.
maxRoutes	Maximum number of routes. Default = 16.
orgPointCode	Originating point code of this node that is passed in the ***opc*** argument. Default = None.
sccpSupport	SCCP support indicator. Default = FALSE.
ituUCICs	Indicator to send and process UCIC messages. Default = FALSE.
qCongOnset1	Queue to the host application congestion level 1 onset. Default = 600.
qCongAbate1	Queue to the host application congestion level 1 abatement threshold. Default = 400.
qCongOnset2	Queue to the host application congestion level 2 onset. Default = 900.
qCongAbate2	Queue to the host application congestion level 2 abatement threshold. Default = 700.
qCongOnset3	Queue to the host application congestion level 3 onset. Default = 1200.
qCongAbate3	Queue to the host application congestion level 3 abatement threshold. Default = 1000.
mCongOnset1	TX percentage of memory remaining congestion level 1 onset. Default = 20.
mCongAbate1	TX percentage of memory remaining congestion level 1 abatement threshold. Default = 25.
mCongOnset2	TX percentage of memory remaining congestion level 2 onset. Default = 10.
mCongAbate2	TX percentage of memory remaining congestion level 2 abatement threshold. Default = 15.
mCongOnset3	TX percentage of memory remaining congestion level 3 onset. Default = 5.
mCongAbate3	TX percentage of memory remaining congestion level 3 abatement threshold. Default = 8.
comLLID	Common language location identifier. Default = NOT_PRESENT.
t18	Time to wait for a response to a group blocking message. value = 12

Field	Description
t19	Time to wait for a response to an initial group blocking message. value = 60 enable = TRUE
t20	Time to wait for a response to a group unblocking message. value= 12 enable = TRUE
t21	Time to wait for a response to an initial group unblocking message. value = 60 enable = TRUE
t22	Time to wait for a response to a circuit group reset message. value = 12
t23	Time to wait for a response to the initial circuit group reset message. value = 60 enable = TRUE
t28	Time to wait for a response to an initial circuit group query message. value = 10 enable = TRUE
tFrstGrpRx	ANSI first group received timer. value = 0
tGrpReset	Group reset timer. value = 1 enable = TRUE
igPassAlng	Ignore pass along messages indicator. Default = FALSE.
extElmts	Extended elements indicator. Default = FALSE.
rawMsgs	Raw messages indicator. Default = FALSE.
	ANSI only. Indicator that the stack should react to first group message if TRUE. Default = FALSE
rmtUsrUnavl	Configures the stack to start in remote user unavailable mode. Default = FALSE.
grpResetEvent	Configures the stack to send up one group reset event instead of many separate circuit reset events. Default = FALSE.
slsFromCICs	Sets the ANSI SLS value to the bottom bits of the CIC. Default = TRUE.
dsblRmtUsrUnavl	Disables appropriate user part test procedure (for SSURN among others). Default = FALSE.
	Restarts T7 when an inbound INR is received (for SSURN among others). Default = FALSE.
disableACL	Disables automatic congestion control (for SSURN among others). Default = FALSE.

Structure members not indicated in the previous table are initialized to appropriate values for proper operation of the ISUP stack and must not be modified.

Example

```
IsupGenParms  cfg;
U32           opc = 1.1.4;

isupInitGenCfg(&cfg, opc);
```

isupInitMgmtAPI

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

Prototype

short **isupInitMgmtAPI** (U8 ***board***, U8 ***srcEnt***)

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

Return values

Return value	Description
ISUP_SUCCESS	
ISUP_BOARD	***board*** is out of range.
ISUP_DRIVER	Error occurred accessing the driver.

Details

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

Example

```
U8 board = 1; /* Tx board number */
U8 srcEnt = MGMT_ENT_ID; /* Entity ID of calling application */

/* open the ISUP management API   */
status = isupInitMgmtAPI( board, MGMT_ENT_ID );
if( status != ISUP_MGMT_SUCCESS )
    {
        printf( "Error: isupInitMgmtAPI [board %d] failed [%d], defaulting to 1\n",
                 board, status );
    }
```

isupInitNSapCfg

Builds a network service access point configuration buffer that can be passed to **isupNSapCfg**.

Prototype

void **isupInitNSapCfg** (IsupNSapCfg ***cfg**, S16 ***switchType***)

Argument	Description

cfg	Pointer to the following IsupNSapCfg structure to initialize:
	`typedef struct _IsupNSapCfg /* ISUP Network Sap Config. struct (LOWER) */`
	`{`
	`  S16        switchType;        /* Protocol Switch                    */`
	`  S16        spId;              /* Service provider id                */`
	`  U8         database[(MAX_DB_LEN+5) & 0xffc];  /* Database name       */`
	`  U8         ssf;               /* Sub service field                  */`
	`  U8         dstEntity;         /* Entity                             */`
	`  U8         dstInstance;       /* Instance                           */`
	`  U8         priority;          /* Priority                           */`
	`  U16        dstProcId;         /* Destination processor id           */`
	`  S16        sapType;           /* Sap type                           */`
	`  U8         route;             /* Route                              */`
	`  U8         numResInd;         /* Max simultaneous res ind to upper  */`
	`  U8         selector;          /* Selector                           */`
	`  U8         fill1;`
	`  MemoryId   mem;               /* Memory region & pool id            */`
	`} IsupNSapCfg;`
	Refer to the Details section for more information.

switchType	Switch type indicator.

Details

After calling **isupInitNSapCfg**, call **isupNSapCfg** to set the network service access point configuration. You can optionally override specific field values before calling **isupNSapCfg**.

The fields of the IsupNSapCfg structure are initialized with the following values:

Field	Description
switchType	One of the following switch type indicators specified in ***switchType***: ISUP_SW_ITU ISUP_SW_ANS88 ISUP_SW_ANS92 ISUP_SW_JNTT
ssf	ISUP_SSF_NAT

Structure members not indicated in the previous table are initialized to appropriate values for proper operation of the ISUP stack and must not be modified.

Example

```
IsupNsapCfg  cfg;
U16          switchType = ISUP_SW_ANS92;

/* Initialize NSAP configuration parameters to default values */

isupInitNSapCfg(&cfg, switchType);
```

isupInitUSapCfg

Builds a user service access point configuration buffer that can be passed to **isupUSapCfg**.

Prototype

void **isupInitUSapCfg** (IsupUSapCfg ***cfg**, S16 ***switchType***)

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td>cfg</td><td>Pointer to the following IsupUSapCfg structure to initialize:
<pre>
{
  S16   switchType;   /* Protocol Switch   */
  U16   fill1;
  U8   database[(MAX_DB_LEN + 5) & 0xffc];   /* database name   */
  U8   wildCardMask[(ISUPADDRLEN + 4) & 0xffc];/* Wild Card Mask */
  Bool   wildCardRoute;   /* Wild Card Routing Flag   */
  Bool   sidInsert;   /* SID insertion Flag   */
  Bool   sidVerify;   /* SID verification Flag   */
  U8   fill2;
  IsupAddr sid;   /* SID   */
  U8   natAddrInd;   /* SID Nature of Address Indicator   */
  U8   sidNumPlan;   /* SID Numbering Plan   */
  U8   sidPresInd;   /* Default presentation indicator   */
  U8   maxMsgLength;   /* Max length of user to user messages   */
  Bool   incSidPresRes;   /* Presentation Restriction of incoming SID */
  Bool   sidPresRes;   /* Presentation Restriction   */
  Bool   reqOpt;   /* Request option   */
  Bool   allowCallMod;   /* Call modification allowed   */
  TimerCfg t1;   /* t1 timer - release sent   */
  TimerCfg t2;   /* t2 timer - suspend received   */
  TimerCfg t5;   /* t5 timer - initial release sent   */
  TimerCfg t6;   /* t6 timer - suspend received   */
  TimerCfg t7;   /* t7 timer - latest address sent   */
  TimerCfg t8;   /* t8 timer - initial address received   */
  TimerCfg t9;   /* t9 timer - latest address sent after ACM */
  TimerCfg t16;   /* t16 timer - reset sent   */
  TimerCfg t17;   /* t17 timer - initial reset sent   */
  TimerCfg t31;   /* t31 timer - call reference frozen period */
  TimerCfg t33;   /* t33 timer - INR sent   */
  TimerCfg tCCR;   /* tCCR timer - continuity recheck timer   */
  TimerCfg t27;   /* t27 timer - waiting for continuity
                             /*   recheck   */
  TimerCfg t34;   /* t34 timer - waiting for continuity   */
                             /* or release message after LoopBackAck   */
  TimerCfg tEx;   /* tEx timer - Exit to be sent   */
  TimerCfg tCRM;   /* Circuit reservation message timer   */
  TimerCfg tCRA;   /* Circuit reservation ack. timer   */
  U8   prior;   /* Priority   */
  U8   route;   /* Route   */
  U8   selector;   /* Selector   */
  U8   fill3;
  MemoryId memory;   /* Memory region & pool id   */

} IsupUSapCfg;
</pre>
Refer to the Details section for more information.</td></tr>
<tr><td>switchType</td><td>Type of switch.</td></tr>
</table>

Details

After calling **isupInitUSapCfg**, call **isupUSapCfg** to set the user service access point configuration. You can optionally override specific field values before calling **isupUSapCfg**.

The fields of the IsupUSapCfg structure are initialized with the following values:

Field	Description
switchType	One of the following switch type indicators specified in ***switchType***: ISUP_SW_ITU ISUP_SW_ANS88 ISUP_SW_ANS92 ISUP_SW_JNTT
sid	Service ID string. BCD string of digits to be inserted into calling party address in outgoing IAM messages. length = 0 string = zero filled
sidInsert	The stack supplies the calling party address (from the sid field) in the outgoing IAM messages. Default = TRUE
sidVerify	The stack verifies that the address passed in the calling party address of outgoing connect requests is the same as that supplied in the sid field. If they are equal, then the screening indicator in the calling party parameter is set to User Provided. If they are not equal, then the screening indicator in the calling party parameter is set to Network Provided, and the calling party address is inserted from the sid field. Default = FALSE The sidVerify field has no effect if the sidInsert field is set to FALSE.
natAddrInd	Nature of address indicator that the ISUP stack inserts into the calling party parameter of outgoing IAM messages if sidInsert is set to TRUE. Default = 0.
sidNumPlan	Numbering plan that ISUP stack inserts into the calling party parameter of outgoing IAM messages if sidInsert is set to TRUE. Default = 0.
sidPresInd	FALSE
incSidPresRes	FALSE
sidPresRes	FALSE
reqOpt	FALSE
allowCallMod	TRUE
maxMsgLength	Maximum length (bytes) of user-to-user data. Default = 20.
t1	Time to wait for a response to a transmitted release message. value = 12 enable = TRUE
t2	Time to wait for a resume message after a suspend message is received. value = 0 enable = FALSE
t5	Time to wait for a response to a transmitted initial release message. value = 60 enable = TRUE

Field	Description
t6	Time to wait for a resume message after a suspend network message is received. value = 30 enable = TRUE
t7	Time to wait for a response (for example, ACM, ANS, CON) to the latest transmitted address message. value = 25 enable = TRUE
t8	Time to wait for a continuity message after receiving an initial address message (IAM) that requires a continuity check. value = 12 enable = TRUE
t9	Time to wait for an answer of an outgoing call after an address complete message (ACM) is received. value = 180 enable = TRUE
t16	Time to wait for an acknowledgement of a transmitted reset message. value = 15 enable = TRUE
t17	Time to wait for an acknowledgement of the initially transmitted reset message. value = 60 enable = TRUE
t27	Time to wait for a continuity check request after a continuity check failure indication is received. value = 240 enable = TRUE
t31	Time to wait before reusing a call reference after a connection is cleared. value = 0 enable = FALSE
t33	Time to wait for a response to a transmitted information request message. value = 15 enable = TRUE
t34	Time to wait for a continuity message (COT) response or a release (REL) response after transmitting a loopback acknowledgment. value = 0 enable = FALSE
tCCR	Continuity recheck timer. value = 0 enable = FALSE
tEx	Time to wait before sending an ANSI exit message. value = 0 enable = FALSE
tCRM	Circuit reservation acknowledgment timer. value = 4 enable = TRUE

Field	Description
tCRA	Time to wait for an IAM message after sending a circuit reservation acknowledgment message. value = 10 enable = TRUE

Structure members not indicated in the previous table are initialized to appropriate values for proper operation of the ISUP stack and must not be modified.

Example

```
IsupUSapCfg  cfg;
U16          swtchType = ISUP_SW_ANS92;

/* Initialize USAP configuration parameters to default values */

isupInitUSapCfg(&cfg, switchType);
```

isupNSapCfg

Sends a network service access point configuration buffer to the TX board and blocks the calling application while waiting for a response.

Prototype

short **isupNSapCfg** (U8 ***board***, IsupNSapCfg ****cfg***, S16 ***spId***)

Argument	Description
board	TX board number.
cfg	Pointer to the IsupNSapCfg structure to send. Refer to *isupInitNSapCfg* on page 154 for the structure definition.
spId	ISUP service access point.

Return values

Return value	Description
ISUP_SUCCESS	
ISUP_BOARD	***board*** is out of range.
ISUP_DRIVER	Error occurred accessing the driver.
ISUP_FAILED	Task on the TX board returned a failure.
ISUP_UNBOUND	Application failed to call **isupInitMgmtAPI** prior to this call.

Details

An application must set the field values in the IsupNSapCfg structure before calling **isupNSapCfg**. Set the values in any of the following ways:

- Call **isupInitNSapCfg** to set the fields to default values.
- Set each field value from within the application.
- Call **isupInitNSapCfg** and then override specific field values before passing the IsupNSapCfg structure to **isupNSapCfg**.

isupNSapCfg is typically called once for each configured network service access point.

Example

```
S16            status;
U8             boardNum = 1;
IsupNSapCfg    cfg;
S16            sapId = 1

/* Populate NSAP configuration structure cfg before calling isupNSapCfg */

if ((status = isupNSapCfg(boardNum, &cfg, sapId)) != ISUP_MGMT_SUCCESS)
{
    printf( "Board %d NSAP %d Send Configuration Information failed: status = %d\n",
        boardNum, sapId, status );
}
else
    printf( "Successfully sent NSAP configuration information for SAP %d on board %d\n",
      sapId, boardNum );
}
```

isupNSapStats

Sends a request to retrieve the statistics for a given network service access point and blocks the calling application while waiting for a response.

Prototype

short **isupNSapStats** (U8 ***board***, S16 ***sapID***, IsupNSapStats ****stats***);

Argument	Description
board	TX board number.
sapId	ISUP service access point.
stats	Pointer to the IsupNSapStats structure where the requested statistics information is returned. For more information, refer to *IsupNSapStats structure* on page 160 and to the Details that follow.

IsupNSapStats structure

```
typedef struct _IsupNSapStats
{
  DateTime dt;              /* Date and time                         */
  Duration dura;            /* Duration                              */
  S32  adrCmpltTx;          /* Address complete transmitted          */
  S32  answerTx;            /* Answer transmitted                    */
  S32  progressTx;          /* Progress transmitted                  */
  S32  contiTx;             /* Continuity transmitted                */
  S32  conChkReqTx;         /* Continuity Check Request transmitted  */
  S32  loopBckAckTx;        /* Lookback Acknowledge transmitted      */
  S32  confusTx;            /* Confusion transmitted                 */
  S32  callModReqTx;        /* Call Modification Request transmitted */
  S32  callModRejTx;        /* Call Modification Reject transmitted  */
  S32  callModComTx;        /* Call Modification Complete transmitted */
  S32  suspTx;              /* Suspend transmitted                   */
  S32  resmTx;              /* Resume transmitted                    */
  S32  forwTx;              /* Forward transmitted                   */
  S32  conTx;               /* Connect transmitted                   */
  S32  relTx;               /* Release transmitted                   */
  S32  overldTx;            /* Overload transmitted                  */
  S32  relCmpltTx;          /* Release Complete transmitted          */
  S32  facTx;               /* Facility Request transmitted          */
  S32  facAckTx;            /* Facility Ack transmitted              */
  S32  facRejTx;            /* Facility Reject transmitted           */
  S32  initAdrTx;           /* Initial Address transmitted           */
  S32  infoTx;              /* Info transmitted                      */
  S32  infoReqTx;           /* Info Request transmitted              */
  S32  passAlongTx;         /* Pass Along transmitted                */
  S32  subsAdrTx;           /* Subsequent Address transmitted        */
  S32  usrToUsrTx;          /* User to User transmitted              */
  S32  uneqCirIdTx;         /* Unequipped Circuit ID transmitted     */
  S32  cirReserveTx;        /* Circuit reservation  transmitted      */
  S32  cirResAckTx;         /* Circuit reservation ack transmitted   */
  S32  exitTx;              /* Exit transmitted                      */
  S32  netResMgmtTx;        /* Network Resourse transmitted          */
  S32  netIdReqTx;          /* Network Id Request transmitted        */
  S32  netIdRspTx;          /* Network Id Request transmitted        */
  S32  chargeTx;            /* Charge transmitted                    */
  S32  loopPrevTx;          /* Loop Prevention transmitted           */
  S32  preRelTx;            /* Pre-Release transmitted               */
  S32  appTransTx;          /* Application Transport transmitted     */
  S32  adrCmpltRx;          /* Address complete received             */
  S32  answerRx;            /* Answer received                       */
  S32  progressRx;          /* Progress received                     */
  S32  contiRx;             /* Continuity received                   */
  S32  conChkReqRx;         /* Continuity Check Request received     */
```

```
  S32  loopBckAckRx;        /* Lookback Acknowledge received          */
  S32  confusRx;            /* Confusion received                     */
  S32  callModReqRx;        /* Call Modification Request received     */
  S32  callModRejRx;        /* Call Modification Reject received      */
  S32  callModComRx;        /* Call Modification Complete received    */
  S32  suspRx;              /* Suspend received                       */
  S32  resmRx;              /* Resume received                        */
  S32  forwRx;              /* Forward received                       */
  S32  conRx;               /* Connect received                       */
  S32  overldRx;            /* Overload received                      */
  S32  relRx;               /* Release received                       */
  S32  relCmpltRx;          /* Release Complete received              */
  S32  facRx;               /* Facility Request received              */
  S32  facAckRx;            /* Facility Ack received                  */
  S32  facRejRx;            /* Facility Reject received               */
  S32  initAdrRx;           /* Initial Address received               */
  S32  infoReqRx;           /* Info Request received                  */
  S32  infoRx;              /* Info received                          */
  S32  passAlongRx;         /* Pass Along received                    */
  S32  subsAdrRx;           /* Subsequent Address received            */
  S32  usrToUsrRx;          /* User to User received                  */
  S32  uneqCirIdRx;         /* Unequipped Circuit ID received         */
  S32  cirReserveRx;        /* Circuit reservation  received          */
  S32  cirResAckRx;         /* Circuit reservation ack received       */
  S32  exitRx;              /* Exit received                          */
  S32  netResMgmtRx;        /* Network Resource received              */
  S32  netIdReqRx;          /* Network Id Request received            */
  S32  netIdRspRx;          /* Network Id Request received            */
  S32  chargeRx;            /* Charge message received                */
  S32  segRx;               /* Segmentation received                  */
  S32  loopPrevRx;          /* Loop Prevention received               */
  S32  preRelRx;            /* Pre-Release received                   */
  S32  appTransRx;          /* Application Transport received         */
} IsupNSapStats;
```

Return values

Return value	Description
ISUP_SUCCESS	
ISUP_BOARD	***board*** is out of range.
ISUP_DRIVER	Error occurred accessing the driver.
ISUP_FAILED	Task on the TX board returned a failure.
ISUP_UNBOUND	Application failed to call **isupInitMgmtAPI** prior to this call.

Details

isupNSapStats automatically resets the statistics counters. Use **isupNSapStatsEx** instead of **isupNSapStats** to control the resetting of the statistics counters. For more information, see *isupNSapStatsEx* on page 163.

Example

```
S16           status;
U8            boardNum=1;
S16           sapId=1;
IsupNsapStats sts;


if ((status = isupNsapStats(boardNum, sapID, &sts)) != ISUP_MGMT_SUCCESS)
{
    printf( "Board %d Service Access Point %d Get Statistics Request failed:
        status = %d\n", boardNum, sapID, status );
}
else
{
    printf("Successfully obtained NSAP statistics information for SAP %d on board %d\n",
        sapID, boardNum);

/* The structure ‘sts’ contains the returned NSAP statistics information */

}
```

isupNSapStatsEx

Sends a request to retrieve the statistics for a given network service access point and blocks the calling application while waiting for a response. This function also lets you control whether statistics counters are reset after function execution.

Prototype

short **isupNSapStats** (U8 ***board***, S16 ***sapID***, IsupNSapStats ****stats*** u8 **reset**)

Argument	Description
board	TX board number.
sapId	ISUP service access point.
stats	Pointer to the IsupNSapStats structure where the requested statistics information is returned. For information, see the Details in *isupNSapStats* on page 160.
reset	Indicates whether to reset the statistics counters after function execution 0 = Reset the statistics counters to zeroes. 1 = Do not reset the statistics counters.

Return values

Return value	Description
ISUP_SUCCESS	
ISUP_BOARD	***board*** is out of range.
ISUP_DRIVER	Error occurred accessing the driver.
ISUP_FAILED	Task on the TX board returned a failure.
ISUP_UNBOUND	Application failed to call **isupInitMgmtAPI** prior to this call.

Example

```
U8              boardNum = 1;
S16             sapId = 1;
IsupNSapStats   sts;
U8              resetStats=1;
S16             status;

if ((status = isupNSapStatsEx(boardNum, sapId, &sts, resetStats)) != ISUP_MGMT_SUCCESS)
{
    printf( "Board %d NSAP %d Statistics Request failed: status = %d\n",
        boardNum, sapId, resetStats, status );
}
else
    printf( "Successfully retrieved NSAP statistics for SAP %d on board %d\n",
        sapId, boardNum );
```

isupResetCircuit

Sends a request to reset the given circuit and blocks the calling application while waiting for a response.

Prototype

short **isupResetCircuit** (U8 ***board***, U32 ***circId***)

Argument	Description
board	TX board number.
circId	Identifier of the circuit to reset.

Return values

Return value	Description
ISUP_SUCCESS	
ISUP_BOARD	***board*** is out of range.
ISUP_DRIVER	Error occurred accessing the driver.
ISUP_FAILED	Task on the TX board returned a failure.
ISUP_UNBOUND	Application failed to call **isupInitMgmtAPI** prior to this call.

Example

```
S16  status;
U8   boardNum=1;
U32  circuitId=1;

if ((status = isupResetCircuit(boardNum, circuitId)) != ISUP_MGMT_SUCCESS)
{       printf( "Board %d Circuit %d Reset Request failed: status = %d\n", boardNum,
          circuitId, status );
}
else
{
    printf( "Successfully reset circuit %d on board %d\n", circuitId, boardNum )
}
```

isupTermMgmtAPI

Closes ISUP management and the channel to the TX board.

Prototype

short **isupTermMgmtAPI** (U8 ***board***)

Argument	Description
board	TX board number.

Return values

Return value	Description
ISUP_SUCCESS	
ISUP_BOARD	***board*** is out of range.
ISUP_UNBOUND	Application failed to call **isupInitMgmtAPI** prior to this call.

Details

Call **isupTermMgmtAPI** to close ISUP management when the application has finished using it.

Example

```
S16             status;
U8              boardNum=1;

if ((status = isupTermMgmtAPI(boardNum)) != ISUP_MGMT_SUCCESS)
{
    printf( "Board %d Close ISUP Management and Channel Request failed: status = %d\n",
        boardNum, status );
}
else
{
    printf( "Successfully closed ISUP Management and Channel on board %d\n",
        boardNum );
}
```

isupTraceControl

Sends a request to control what is saved to the trace log and blocks the calling application while waiting for a response.

Prototype

short **isupTraceControl** (U8 ***board***, Bool ***onOff***, U32 ***flags***)

Argument	Description
board	TX board number.
onOff	Tracing is on if non-zero. Tracing is off if zero.
flags	Bit map (set of bits) that indicates what type of tracing is performed. Refer to the Details section for more information.

Return values

Return value	Description
ISUP_SUCCESS	
ISUP_BOARD	***board*** is out of range.
ISUP_DRIVER	Error occurred accessing the driver.
ISUP_FAILED	Task on the TX board returned a failure.
ISUP_UNBOUND	Application failed to call **isupInitMgmtAPI** prior to this call.

Details

Supported values for types of tracing that can be combined using the OR operation into ***flags*** include:

Value	Description
TRACE_DATA	Protocol data tracing.
TRACE_EVENT	Event tracing.
ENCODE_ERROR	Alarm of message encode errors.
ENUM_WARNING	Alarm of enumeration warnings. ISUP checks values for most fields while encoding or decoding messages. An enumeration warning is generated when a field value is encountered that is not explicitly defined in the protocol specification for the configured switch type.

Example

```
S16    status;
U8     board = 1;

if ((status = isupTraceControl(board, 1, TRACE_DATA | TRACE_EVENT))
      != ISUP_MGMT_SUCCESS)
    printf( "Board %d Trace Control Request failed: status = %d\n",
       board, status );
}
```

isupUnblockCircuit

Sends a request to unblock the given circuit and blocks the calling application while waiting for a response.

Note: This function is not supported in BICC. The ISUP stack layer reports a service unavailable alarm message if this function is invoked for BICC.

Prototype

short **isupUnblockCircuit** (U8 ***board***, U32 ***circId***)

Argument	Description
board	TX board number.
circId	Identifier of the circuit to unblock.

Return values

Return value	Description
ISUP_SUCCESS	
ISUP_BOARD	***board*** is out of range.
ISUP_DRIVER	Error occurred accessing the driver.
ISUP_FAILED	Task on the TX board returned a failure.
ISUP_UNBOUND	

Example

```
S16  status;
U8   boardNum=1;
U32  circuitId=1;

if ((status = isupUnblockCircuit(boardNum, circuitId)) != ISUP_MGMT_SUCCESS)
{
    printf( "Board %d Circuit %d Unblock Request failed: status = %d\n", boardNum,
        circuitId, status );
}
else
{
    printf( "Successfully unblocked circuit %d on board %d\n", circuitId, boardNum );
}
```

isupUSapCfg

Sends a user service access point configuration buffer to the TX board and blocks the calling application while waiting for a response.

Prototype

short **isupUSapCfg** (U8 ***board***, IsupUSapCfg ****cfg***, U16 ***sapId***)

Argument	Description
	TX board number.
cfg	Pointer to the IsupUSapCfg structure to send. Refer to *isupInitUSapCfg* on page 155 for the structure definition.
sapId	ISUP service access point.

Return values

Return value	Description
ISUP_SUCCESS	
ISUP_BOARD	***board*** is out of range.
ISUP_DRIVER	Error occurred accessing the driver.
ISUP_FAILED	
ISUP_UNBOUND	

Details

An application must set the field values in the IsupUSapCfg structure before calling **isupUSapCfg**. Set the values in any of the following ways:

- Call **isupInitUSapCfg** to set the fields to default values.
- Set each field value from within the application.
- Call **isupInitUSapCfg** and then override specific field values before passing the IsupUSapCfg structure to **isupUSapCfg**.

isupUSapCfg is typically called once for each configured user service access point.

Example

```
S16            status;
U8             boardNum = 1;
IsupUSapCfg    cfg;
S16            sapId = 1;

/* Populate USAP configuration structure cfg before calling isupUSapCfg */

if ((status = isupUSapCfg(boardNum, &cfg, sapId)) != ISUP_MGMT_SUCCESS)
{
    printf( "Board %d NSAP %d Send Configuration Information failed: status = %d\n",
        boardNum, sapId, status );
}
else
    printf( "Successfully sent USAP configuration information for SAP %d on
       board %d\n", sapId, boardNum );
}
```

isupValidateCircuit

Sends a request to validate the given circuit and blocks the calling application while waiting for a response.

Note: This function is not supported in BICC. If this function is invoked for BICC, the ISUP stack layer reports a service unavailable alarm message.

Prototype

short **isupValidateCircuit** (U8 ***board***, U32 ***circId***)

Argument	Description
board	TX board number.
circId	Identifier of the circuit to validate.

Return values

Return value	Description
ISUP_BOARD	***board*** is out of range.
ISUP_DRIVER	Error occurred accessing the driver.
ISUP_FAILED	
ISUP_UNBOUND	Application failed to call **isupInitMgmtAPI** prior to this call.

Example

```
S16  status;
U8   boardNum=1;
U32  circuitId=1;

if ((status = isupValidateCircuit(boardNum, circuitId)) != ISUP_MGMT_SUCCESS)
{
    printf( "Board %d Circuit %d Validate Request failed: status = %d\n", boardNum,
        circuitId, status );
}
else
{
    printf( Successfully validated circuit %d on board %d\n", circuitId, boardNum );
}
```

8 Demonstration programs and utilities

Summary of the demonstration programs and utilities

The ISUP service provides the following demonstration programs and utilities:

Program	Description
isupcfg	Downloads the ISUP configuration to the TX board at boot time.
isupmgr	Monitors and manages the status of the ISUP layer.
term	Demonstrates how the ISUP service accepts an incoming call from the specified TX board.
orig	Demonstrates how the ISUP service generates an outbound call to the specified TX board.

ISUP configuration utility: isupcfg

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

Usage

```
isupcfg options
```

Requirements

- A computer with a TX board installed
- Windows or UNIX
- NaturalAccess™
- NaturalAccess™ SS7

Procedure

To run *isupcfg*:

Step	Action
1	From the command line prompt, navigate to the *\Program Files\Dialogic\tx\bin* directory under Windows or the */opt/dialogic/tx/bin/* directory under UNIX.
2	Enter the following command: `isupcfg options` where ***options*** include: **Options** / **Description**: -b ***board*** — Board number to which the ISUP configuration is downloaded. Default = 1. -f ***filename*** — Name and location of the ISUP configuration file to be downloaded. The ISUP 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.

Options	Description
-b ***board***	Board number to which the ISUP configuration is downloaded. Default = 1.
-f ***filename***	Name and location of the ISUP configuration file to be downloaded.

Details

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

ISUP layer status: isupmgr

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

Usage

```
isupmgr -b board
```

Requirements

- A computer with a TX board installed
- Windows or UNIX
- NaturalAccess™
- NaturalAccess™ SS7

Procedure

To run *isupmgr*:

Step	Action
1	From the command line prompt, navigate to the *\Program Files\Dialogic\tx\bin* directory under Windows or the */opt/dialogic/tx/bin/* directory under UNIX.
2	Enter the following command: `isupmgr -b board` where ***board*** is the TX board number on which the ISUP layer is loaded.

The *isupmgr* program supports the following commands:

Command	Description
VALIDATE ***number***	Validates the specified circuit.
RESET ***number***	Resets the specified circuit.
BLOCK ***number***	Blocks the specified circuit.
UNBLOCK ***number***	Unblocks the specified circuit.
DELETE ***number***	Deletes the specified circuit.
STATUS ***number***	Displays status of specified circuit.
STATS ***entity number***	Retrieves statistics for the given ***entity*** (either NSAP or CIRCUIT).
GET ***entity number***	Retrieves configuration information for the given ***entity*** (either GEN, USAP, NSAP, or CIRCUIT).
TRACE ON | OFF	Turns buffer tracing ON or OFF for the following parameters: Event - Traces events between the ISUP task and MTP and between the ISUP task and the application. Data - Traces the data to and from the MTP. Error - Traces errors. Default is ON. Warning - Traces lower level warnings. Element - Traces each information element during encoding and decoding. Token - Traces each token during encoding and decoding. Timer - Traces timer starts, stops, and pops.
BOARD ***board***	Switches to a new target board (***board***).
QUIT	Quits the application.

Details

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

Accepting incoming calls: term

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

Usage

```
term options
```

Requirements

- A computer with a TX board installed
- Windows or UNIX
- NaturalAccess™
- NaturalAccess™ SS7

Procedure

To run *term*:

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>From the command line prompt, navigate to the \Program Files\Dialogic\tx\src\isup\samples directory under Windows or the /opt/dialogic/tx/src/isup/samples/ directory under UNIX.</td></tr>
<tr><td>2</td><td>Enter the following command:
<code>term options</code>
where options include:
<table>
<tr><th>Options</th><th></th></tr>
<tr><td>-b board</td><td>TX board number on which to bind. Default = 1.</td></tr>
<tr><td>-i</td><td>ITU switch type.</td></tr>
<tr><td>-c number</td><td>Circuit ID with which to test. Default = 1.</td></tr>
<tr><td>-s</td><td>BICC switch type.</td></tr>
</table></td></tr>
<tr><td>3</td><td>Press any key to exit the program.</td></tr>
</table>

Details

The ISUP termination sample program prints out all events retrieved by **ISUPRetrieveMessage**, including blocks, unblocks, and resets received.

Generating outbound calls: orig

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

Usage

```
orig options
```

Requirements

- A computer with a TX board installed
- Windows or UNIX
- NaturalAccess™
- NaturalAccess™ SS7

Procedure

To run *orig*:

<table>
<tr><th>Step</th><th>Action</th></tr>
<tr><td>1</td><td>From the command line prompt, navigate to the \Program Files\Dialogic\tx\src\isup\samples directory under Windows or the /opt/dialogic/tx/src/isup/samples/ directory under UNIX.</td></tr>
<tr><td>2</td><td>Enter the following command:
<code>orig options</code>
where options include:
<table>
<tr><th>Options</th><th>Description</th></tr>
<tr><td>-i</td><td>ITU switch type.</td></tr>
<tr><td>-b board</td><td>TX board number on which to bind. Default = 1.</td></tr>
<tr><td>called number</td><td>Phone number to which the call is placed.</td></tr>
<tr><td>circuit index</td><td></td></tr>
<tr><td>calling number</td><td>Phone number from which the call is placed.</td></tr>
<tr><td>-s</td><td>BICC switch type.</td></tr>
</table></td></tr>
<tr><td>3</td><td>Press r to release the call, and press q to quit the program.</td></tr>
</table>

Details

The ISUP origination sample program prints out all events retrieved by **ISUPRetrieveMessage**, including blocks, unblocks, and resets received.

9 Tokens and events reference

Usage information overview

This section provides the detailed encoding reference for:

- Events (messages) passed between the application and the SS7 ISUP service layer
- Information elements (IEs) and tokens that compose the events

Data structures

C-language definitions for all token structures, information element structures, event structures, and associated constants are provided in the *isupmsgs.h* file. For information about the information elements used for generating specific ISUP messages for each switch type, refer to *Sending ISUP protocol messages* on page 309 and *Receiving ISUP protocol messages* on page 312.

Coding of presence indicators

Each token within an information element and 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:

```
#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       */
```

The definitions for the presence indicators are included in the *isupmsgs.h* file.

Tokens

This topic specifies the format of tokens, or fields, found within the information elements that make up the SS7 ISUP events passed between the application and the ISUP layer.

<table>
<tr><th>Token</th><th>Type</th></tr>
<tr><td>U8</td><td>A quantity specified in 8 bits:
<pre>
typedef struct tknU8 /* token U8   */
{
    U8 pres;   /* present flag   */
    U8 val;   /* value   */
    U16 spare1;   /* for alignment   */
} TknU8;
</pre></td></tr>
<tr><td>U16</td><td>A quantity specified in 16 bits:
<pre>
typedef struct tknU16   /* token U16   */
{
    U8 pres;   /* present flag */
    U8 spare1;   /* for alignment */
    U16 val;   /* value   */
} TknU16;
</pre></td></tr>
<tr><td>U32</td><td>A quantity specified in 32-bits:
<pre>
typedef struct tknU32   /* token U32   */
{
    U8 pres;   /* present flag */
    U8 spare1;   /* for alignment */
    U16 spare2;   /* for alignment */
    U32 val;   /* value   */
} TknU32;
</pre></td></tr>
<tr><td>String</td><td>A variable length sequence of octets such as an address (sequence of digits):
<pre>
typedef struct tknStr   /* token string   */
{
    U8 pres;   /* present flag   */
    U8 len;   /* length   */
    U16 spare1;   /* for alignment */
    U8 val[(MF_SIZE_TKNSTR + 3) & 0xffc];
                          /* string value   */
} TknStr;
</pre></td></tr>
<tr><td>Extended</td><td>The presence indicator for the entire token, a type field, a length field, and the data itself. This token exists only in the extended element structure.
<pre>
typedef struct _siTknExt /* token string   */
{
    U8 pres;   /* present flag   */
    U8 type;   /* type value   */
    U8 len;   /* length   */
    U8 spare1;   /* for alignment */
    U8 val[MF_SIZE_EXTTKN];
} TknExt;
</pre>
The type, len, and val fields are encoded inside the ISUP packet after all known optional parameters.</td></tr>
</table>

Events

This topic specifies the layout of the following events that are passed between the application and the ISUP layer implementation:

- Connect
- Connect Status
- Information
- Resume
- Status
- Release
- Suspend
- Facility
- Raw ISUP packet

Connect

The ISUP Connect event structure is populated by the application when a circuit switch connection is requested. Information elements specific to the Japan/NTT variant are in **bold**.

```
typedef struct _siConEvnt      /* ISUP Connect Event                    */
{
    SiNatConInd      natConInd;      /* Nature of connection indicators      */
    SiFwdCallInd     fwdCallInd;     /* forward call indicators              */
    SiCgPtyCat       cgPtyCat;       /* calling party category               */
    SiTxMedReq       txMedReq;       /* transmission medium requirement      */
    SiTxMedReq       txMedReqPr;     /* transmission medium requirement prime */
    SiCdPtyNum       cdPtyNum;       /* called party number                  */
    SiTranNetSel     tranNetSel;     /* transit network selection            */
    SiCallRef        callRef;        /* call reference                       */
    SiCgPtyNum       cgPtyNum;       /* calling party number                 */
    SiOpFwdCalInd    opFwdCalInd;    /* optional forward call indicators     */
    SiRedirNum       redirgNum;      /* redirection number                   */
    SiRedirInfo      redirInfo;      /* redirection information              */
    SiCugIntCode     cugIntCode;     /* closed grp interlock code            */
    SiConnReq        connReq;        /* connection request                   */
    SiOrigCdNum      origCdNum;      /* original called number               */
    SiUsr2UsrInfo    usr2UsrInfo;    /* user to user information             */
    SiAccTrnspt      accTrnspt;      /* access transport                     */
    SiAppTransParam  appTransParam;  /* application transport as per         */
                                     /*   BICC req: Q.1902.3                 */
    SiChargeNum      chargeNum;      /* connected number                     */
    SiOrigLineInf    origLineInf;    /* originating line info                */
    SiUsrServInfo    usrServInfo;    /* user service information             */
    SiUsr2UsrInd     usr2UsrInd;     /* user to user indicators              */
    SiPropDly        propDly;        /* propagation delay counter            */
    SiUsrServInfo    usrServInfo1;   /* user service info prime              */
    SiNetSpecFacil   netFac;         /* network specific facility            */
    SiSigPointCode   orgPteCde;      /* originating ISC pnt code             */
    SiGenDigits      genDigits;      /* generic digits                       */
    SiGenNum         genNmb;         /* generic number                       */
    SiRemotOper      remotOper;      /* remote operations                    */
    SiParmCompInfo   parmCom;        /* parameter compatibility information  */
    SiNotifInd       notifInd;       /* notification indicator               */
    SiInfoInd        infoInd;        /* information indicator                */
    SiServiceAct     serviceAct;     /* service activation                   */
    SiMlppPrec       mlppPrec;       /* MLPP precedence                      */
    SiTxMedReq       txMedUsed;      /* transmission medium used             */
    SiBckCalInd      bckCallInd;     /* backward call indicators             */
    SiOptBckCalInd   optBckCalInd;   /* optional backward call indicators    */
```

```
    SiConnectedNum   connNum;         /* connected number                       */
    SiAccDelInfo     accDelInfo;      /* access delivery info                   */
    SiPropDly        cllHstry;        /* call history information               */
    SiRedirNum       redirNum;        /* redirection number                     */
    SiRedirRestr     redirRstr;       /* redirection restriction                */
    SiBusinessGrp    businessGrp;     /* business group                         */
    SiCarrierId      carrierId;       /* carrier identification                 */
    SiCarrierSelInf  carSelInf;       /* carrier selection info                 */

    SiEgress         egress;          /* egress service                         */
    SiGenAddr        genAddr;         /* generic address                        */
    SiInfoReqInd     infoReqInd;      /* info request indicators                */
    SiJurisInf       jurisInf;        /* jurisdiction information               */
    SiNetTransport   netTransport;
                                      /* network transport                      */
    SiSpecProcReq    specProcReq;     /* special processing req.                */
    SiTransReq       transReq;        /* transaction request                    */
    SiEchoCtl        echoControl;     /* echo control                           */
    SiCirAssignMap   cirAssignMap;
                                      /* circuit assignment map                 */
    SiGenName        genName;         /* generic name                           */
    SiHopCount       hopCount;        /* hop counter                            */
    SiOpServInfo     opServInfo;      /* operator services information          */
    SiServiceCode    serviceCode;     /* service code                           */
    SiLocNum         locNum;          /* location number                        */
    SiMsgAreaInfo    msgAreaInfo;     /* message area information               */
    SiContractorNum  contractorNum;   /* contractor number                      */
    SiCgNumNonNotRsn cgNumNonNotRsn;
                                      /* calling number non-notification reason*/
    SiAddUsrId       addUsrId;        /* additional user identification         */
    SiCarrierInfoTrans carrierInfoTrans;
                                      /* carrier information transfer           */
    SiCCSS           cCSS;
    SiNetMngmtCtrls  netMngmtCtrls;
    SiCirAssMap      cirAssMap;
    SiCallDivTrtmnt  callDivTrtmnt;
    SiCdINNum        cdINNum;
    SiCallOffTrtmnt  callOffTrtmnt;
    SiConfTrtmnt     confTrtmnt;
    SiUIDCapInd      uIDCapInd;
    SiCollCallReq    collCallReq;
    SiBckGVNS        bckGVNS;
    SiFreePhnInd     freePhone;
    SiScfID          scfID;
    SiCorrelationID  corrId;
    SiElementExt     elementExt[NUM_EXT_ELMTS];
} SiConEvnt;
```

Connect Status

The Connect Status event structure is filled by the application to signal connection status information to the far exchange while the connection is established. Connection status information includes address complete, progress, information request, and general information. Information elements specific to the Japan/NTT variant are in **bold**.

```
typedef struct _siCnStEvnt       /* Connect Status Event                */
{
    SiSubNum         subNum;          /* subsequent number                    */
    SiBckCalInd      bckCallInd;      /* backward call indicators             */
    SiChargeNum      chargeNum;       /* connected number                     */
    SiOptBckCalInd   optBckCalInd;    /* optional backward call indicators    */
    SiCauseDgn       causeDgn;        /* cause indicators                     */
    SiConnectedNum   connNum;         /* connected number                     */
    SiUsr2UsrInd     usr2UsrInd;      /* user to user indicators              */
    SiUsr2UsrInfo    usr2UsrInfo;     /* user to user information             */
    SiRedirInfo      redirInfo;       /* redirection information              */
    SiAccTrnspt      accTrnspt;       /* access transport                     */
    SiAppTransParam  appTransParam;   /* application transport as per        */
```

```
                                  /*   BICC req: Q.1902.3                */
    SiCalModInd      calModInd;      /* call modification indicators      */
    SiEvntInfo       evntInfo;       /* event information                 */
    SiRedirNum       redirNum;       /* redirection number                */
    SiInfoInd        infoInd;        /* info indicators                   */
    SiInfoReqInd     infoReqInd;     /* info request indicator            */
    SiRedirNum       redirgNum;      /* redirecting number                */
    SiCgPtyCat       cgPtyCat;       /* calling party category            */
    SiCgPtyNum       cgPtyNum;       /* calling party number              */
    SiIndex          index;          /* index                             */
    SiConnReq        connReq;        /* connection request                */
    SiCallRef        callRef;        /* call reference                    */
    SiNotifInd       notifInd;       /* notification indicator            */
    SiTxMedReq       txMedUsed;      /* transmission medium used          */
    SiEchoCtl        echoControl;    /* echo control                      */
    SiAccDelInfo     accDelInfo;     /* access delivery information       */
    SiGenNum         genNmb;         /* generic number                    */
    SiGenDigits      genDigits;      /* generic digits                    */
    SiParmCompInfo   parmCom;        /* parameter compatibility information */
    SiCllDiverInfo   cllDivr;        /* call Diversion information        */
    SiNetSpecFacil   netFac;         /* network specific facility         */
    SiRemotOper      remotOper;      /* remote operations                 */
    SiServiceAct     serviceAct;     /* service activation                */
    SiRedirRestr     redirRstr;      /* redirect restriction              */
    SiMcidReqInd     mcidReq;        /* MCID request indicators           */
    SiMcidRspInd     mcidRsp;        /* MCID response indicators          */
    SiMsgCompInfo    msgCom;         /* msg compatibility information     */
    SiOrigLineInf    origLineInf;    /* originating line information      */
    SiBusinessGrp    businessGrp;    /* business group                    */
    SiNetTransport   netTransport;   /* network transport                 */
    SiMsgAreaInfo    msgAreaInfo;    /* message area information          */
    SiChargeInfo     chargeInfo;     /* charge information                */
    SiChargeInfoType chargeInfoType; /* charge information type           */
    SiChargeInfoDly  chargeInfoDly;  /* charge information delay          */
    SiCarrierInfoTrans carrierInfoTrans; /* carrier information transfer   */
    SiConfTrtmnt   confTrtmnt;
    SiUIDActionInd uIDActionInd;
    SiCallXferNum  callXferNum;
    SiBckGVNS      bckGVNS;
    SiCCNR         cCNR;
    SiElementExt   elementExt[NUM_EXT_ELMTS];
} SiCnStEvnt;
```

Information

The Information event structure is populated while sending user-to-user information associated with an established connection to the far party.

```
 typedef struct _siInfoEvnt      /* Information Event        */
{
    SiCallRef     callRef;       /* call reference           */
    SiPassAlng    passAlng;      /* pass along               */
    SiUsr2UsrInfo usr2UsrInfo;   /* user to user information */
    SiAccTrnspt   accTrnspt;     /* access transport         */
    SiElementExt  elementExt[NUM_EXT_ELMTS];
} SiInfoEvnt;
```

Resume

The Resume event structure indicates that a suspended connection has resumed and sends a RESUME message to the far exchange.

```
typedef struct _siResmEvnt     /* resume event              */
{
    SiSusResInd  susResInd;    /* Suspend/Resume indicators */
    SiCallRef    callRef;      /* call reference            */
    SiElementExt elementExt[NUM_EXT_ELMTS];
} SiResmEvnt;
```

Status

The Status event structure is populated while a global or circuit specific status message is sent to the far exchange.

```
typedef struct _siStaEvnt         /* Status Event */
{
    iRangStat            rangStat;        /* range and status                          */
    iCirGrpSupMTypInd    cgsmti;          /* circuit grp. supervision msg. type ind. */
    SiCirStateInd        cirStateInd;     /* circuit state indicators                  */
    SiContInd            contInd;         /* continuity indicator                      */
    SiCauseDgn           causeDgn;        /* cause indicators                          */
    SiParmCompInfo       parmCom;         /* parameter compatibility information      */
    SiNatConInd          natConInd;       /* Nature of connection indicators          */
    SiCirAssignMap       cirAssignMap;    /* circuit assignment map                    */
    SiMsgCompInfo        msgCom;          /* message compatibility information        */
    SiAppTransParam      appTransParam;   /* application transport as per             */
                                          /*  BICC req: Q.1902.3                      */
    SiOptBckCalInd       optBckCalInd;    /* optional backward call indicators        */
    SiOpFwdCalInd        opFwdCalInd;     /* optional forward call indicators         */
    SiCallXferRef        callXferRef;
    SiLoopPrevInd        loopPrevInd;
    SiElementExt         elementExt[NUM_EXT_ELMTS];  /* extended elements */
} SiStaEvnt;
```

Release

The Release event structure is populated while a circuit switched connection is released. The associated event generates a RELEASE message to the far exchange.

```
typedef struct _siRelEvnt        /* Release Event                       */
{
    SiCauseDgn     causeDgn;     /* cause indicators                    */
    SiRedirInfo    redirInfo;    /* redirection information             */
    SiRedirNum     redirNum;     /* redirection number                  */
    SiRedirgNum    redirgNum;    /* redirection number                  */
    SiCallRef      callRef;      /* call reference                      */
    SiCugIntCodeA  cugIntCodeA;  /* closed group interlock code         */
    SiSigPointCode sigPointCode; /* signalling point code               */
    SiAccTrnspt    accTrnspt;    /* access transport                    */
    SiUsr2UsrInfo  usr2UsrInfo;  /* user to user information            */
    SiAutoCongLvl  autoCongLvl;  /* auto congestion level               */
    SiAccDelInfo   accDelInfo;   /* access delivery information         */
    SiParmCompInfo parmCom;      /* parameter compatibility information */
    SiNetSpecFacil netFac;       /* network specific facility           */
    SiRedirRestr   redirRstr;    /* redirection restriction             */
    SiUsr2UsrInd   usr2UsrInd;   /* user to user indicators             */
    SiChargeNum    chargeNum;    /* charge number                       */
    SiGenAddr      genAddr;      /* generic address                     */
    SiServiceAct   serviceAct;   /* service activation                  */
    SiElementExt   elementExt[NUM_EXT_ELMTS];
} SiRelEvnt;
```

Suspend

The Suspend event structure sends a suspend request to the far exchange and the associated API call starts timer T2. If no resume or release is received before timer T2 expires, the circuit is released. A release indication is sent to the application and an REL message is sent to the far exchange.

```
typedef struct _siSuspEvnt     /* suspend event             */
{
    SiSusResInd  susResInd;    /* suspend/resume indicators */
    SiCallRef    callRef;      /* call reference            */
    SiElementExt elementExt[NUM_EXT_ELMTS];
} SiSuspEvnt;
```

Facility

The Facility event structure is populated while a facility request is sent to the far exchange.

```
typedef struct _siFacEvnt      /* facility event                 */
{
    SiFacInd       facInd;     /* facility indicator             */
    SiFacInfInd    facInfInd;  /* facility information indicator */
    SiCdPtyNum     cdPtyNum;   /* called party number            */
    SiCgPtyNum     cgPtyNum;   /* calling party number           */
    SiCallRef      callRef;    /* call reference                 */
    SiUsr2UsrInd   usr2UsrInd; /* user to user indicator         */
    SiCallRef      callRef;    /* call reference                 */
    SiCauseDgn     causeDgn;   /* cause indicator                */
    SiMsgCompInfo  msgCom;     /* msg compatibility info         */
    SiParmCompInfo parmCom;    /* param compatibility info       */
    SiRemotOper    remotOper;  /* remote operations              */
    SiServiceAct   serviceAct; /* service activation             */
    SiCallXferNum  callXferNum;
    SiAccTrnspt    accTrans;
    SiNotifInd     notifInd;
    SiElementExt   elementExt[NUM_EXT_ELMTS];
} SiFacEvnt;
```

Raw ISUP packet

Use the Raw ISUP event structure to send raw ISUP message data to the far exchange.

```
typedef struct _siRawEvnt        /* Raw message  */
{
    U8 length;
    U8 data[MAX_ISUP_PACKET];
} SiRawEvnt;
```

10 Information elements reference

Information elements overview

This section specifies the layout of the information elements (IEs) that comprise the events passed between the application and the SS7 ISUP layer implementation. Each topic in this section contains a description and structure definition for an IE, followed by a table that lists the tokens contained within the respective structure. The tables are formatted using the following standards:

- Each column within the table specifies tokens for a specific protocol variant. If a variant does not support the IE, that variant is not shown in the table. For BICC variants, the ANSI BICC follows ANSI variants and ITU BICC follows ITU variants with the few exceptions that are mentioned in the BICC specifications. In this section, all tables specify only the ITU BICC variant.
- An asterisk (*) indicates the token is supported by the specified protocol variant.
- A blank cell indicates the token is not supported by the specified protocol variant.
- The bit positions represented by spare and reserved tokens are shown, where applicable.

Element header

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

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

Extended element

Extended elements enable an application to send IEs that are proprietary or unknown to SS7 ISUP. These extended elements are passed as optional parameters in message types that support optional parameters in them. For example, the IAM message supports optional parameters, so any elements in the extended element fields are encoded into the message.

The RSC (reset circuit) message does not support optional parameters, so no extended elements can be passed in that event structure. Passing extended elements for a message that does not support optional parameters causes the message transmission to fail.

On the receive side of applications, the same event structures with extended elements are returned by **ISUPRetrieveMessage**. The application can choose to ignore the extended elements.

Note: To transmit or receive extended elements, configure the ISUP task to allow extended elements. For information, see the *Dialogic® NaturalAccess™ Signaling Software Configuration Manual*.

The extended element consists of the standard element header and an extended token:

```
typedef struct siElementExt
{
    ElmtHdr eh;      /* element header       */
    TknExt  tknExt;  /* extended information */
} SiElementExt;
```

Access delivery

Associated variants: BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Contains information sent in the backward direction to indicate that a setup message was sent to the destination address.

```
typedef struct _accDelInfo  /* Access delivery information */
{
    ElmtHdr eh;             /* element header              */
    TknU8   delInd;         /* delivery indicator          */
    TknU8   reserved;       /* reserved for future use     */
} SiAccDelInfo;
```

The delInd field is encoded to one of the following values:

0x00	A setup message was generated.
0x01	No setup message was generated.

Tokens

Token	BICC	ETSI V2	ETSI V3	ITU White	ITU 97
delInd	*	*	*	*	*
reserved	B-H	B-H	B-H	H-H	B-H

Access transport

Associated variants: All

Contains one or more Q.931 information elements passed through transparently to the far exchange or the CPE. The Q.931 IEs contained in the infoElmts field must be encoded or decoded by the application as specified in Q.931. Refer to ANSI T1.607 or ITU-T Q.763 for more information.

```
typedef struct _accTrnspt  /* Access transport     */
{
    ElmtHdr eh;            /* element header       */
    TknStr  infoElmts;     /* Information elements */
} SiAccTrnspt;
```

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	ANSI 92	ANSI 95	BICC	ETSI V2	ETSI V3
infoElmts	*	*	*	*	*	*

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White	ITU 97	Q.767
infoElmts	*	*	*	*

Application transport parameter

Associated variants: BICC, ETSI V2, ETSI V3, ITU 97

Allows the peer-to-peer communication of the application transport mechanism for ISUP user applications. This IE is encoded as part of an extended element for ETSI V2, ETSI V3 and ITU97. For more information, refer to *Extended element* on page 185.

For BICC, the existing scheme is enhanced to allow one application transport parameter IE in a call setup message (ACM, ANM , CPG, CON, IAM, or PRI) or an application transport message (APM). This IE is encoded as an optional parameter for transporting bearer-related information between peer-to-peer BICC nodes.

If the APM user application needs to encode multiple application IEs for different sequences, then it can encode them in extended elements. The application is responsible for following the rules of encapsulated application information segmentation and segment re-assembly specified in Q.765. The application developer must restrict the length of the message to 272 bytes (limit specified by MTP3) while encoding the APM user information that contains the data.

In a typical IAM message, the mandatory and optional IEs occupy approximately 50 - 60 bytes or octets. This requires the application to restrict APM user information to approximately 180 bytes or octets including the APM header information.

For example, the typical IAM message includes CIC, Message Type, Nature of Indicators, Forward Call Indicators, Calling Party Category, Transmission Medium Requirement, Called Party Number, Calling Party Number, Redirecting Number, and so on.

This information is presented transparently to the application as a character array, as specified in Q.1990 and Q.1970.

```
typedef struct _appTransParam /* Application transport parameter (APP) */
{
    ElmtHdr eh;               /* element header */
    TknU16  appContextInd;    /* application context indicator (ACI) */
    TknU8   relCallInd;       /* release call indicator (RCI) */
    TknU8   sendNotifInd;     /* send notification indicator */
    TknU8   spare;
    TknU8   apmSegInd;        /* APM segmentation indicator */
    TknU8   seqInd;           /* sequence indicator */
    TknU8   segLocRef;        /* segmentation local reference */
    TknStrE apmUserInfo;      /* APM user information field */
                              /* (token string size [255]) -- the data  to be filled  */
                              /* according to Q.1990 and Q.1970 */
} SiAppTransParam;
```

The fields in the SiAppTransParam are encoded as follows:

Field	Description
appContextInd	Application context indicator values for APM 98 and APM 2000 user applications. Defined values: 0x00 = APP_ACI_UCEH_ASE Unidentified context and Error Handling (UCEH) ASE, used by the APM 98 user application. 0x01 = APP_ACI_PSS1_ASE PSS1 ASE (VPN), used by the APM 98 user application. 0x03 = APP_ACI_CHARGING_ASE Charging ASE, used by the APM 98 user application. 0x04 = APP_ACI_GAT_ASE GAT ASE, used by the APM 98 and APM 2000 user applications. 0x05 = APP_ACI_BAT_ASE BAT ASE, used by the APM 98 and APM 2000 user applications. 0x06 = APP_ACI_EUCEH_ASE Enhanced Unidentified context and Error Handling ASE (EUCEH ASE), used by the APM 98 and APM 2000 user applications.
relCallInd	Release call indicator (RCI). Defined values: 0x00 = APP_DONT_RELCALL Do not release call. 0x01 = APP_RELCALL Release call.
sendNotifind	Send notification indicator (SNI). Defined values: 0x00 = APP_DONT_SENDNOTIF Do not send notification.
apmSegInd	APM segmentation indicator. Defined value: 0x00 = APP_APM_SEGIND_FINAL_SEGM Indicator for the final segment. For the number of following segments, use the decimal value.
seqInd	Sequence indicator (SI). Defined values: 0x00 = APP_SUBSEG_TO_FIRST Subsequent segment to first segment. 0x01 = APP_NEWSEQUENCE New sequence.
	Segmentation local reference.
apmUserInfo	APM user information field, which contains the data to be transported.

Tokens

Token	BICC
appContextInd	*
relCallInd	*
sendNotifind	*
spare	2(3-7)
apmSegInd	*
seqInd	*
segLocRef	*
apmUserInfo	*

Automatic congestion level

Associated variants: All except ANSI 88

A level of congestion exists at the sending exchange.

```
typedef struct _autoCongLvl /* Automatic Congestion Level */
{
    ElmtHdr eh;             /* element header             */
    TknU8   auCongLvl;      /* auto congestion level      */
} SiAutoCongLvl;
```

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

0x01	ACLVL_LVL1	Congestion level 1 exceeded
0x02	ACLVL_LVL2	Congestion level 2 exceeded
0x03	ACLVL_LVL3	Congestion level 3 exceeded; spare in BICC

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 92		BICC	ETSI V2	ETSI V3
auCongLvl	*	*	*	*	*

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White	ITU 97	Q.767
auCongLvl	*	*	*	*

Backward call indicators

Associated variants: All

Contains information sent in the backward direction to enable the originating exchange to complete processing a call.

```
typedef struct _bckCalInd    /* Backward Call Indicators      */
{
    ElmtHdr eh;              /* element header                */
    TknU8   chrgInd;         /* Charge Indicator              */
    TknU8   cadPtyStatInd;   /* called party status ind.      */
    TknU8   cadPtyCatInd;    /* called party category ind     */
    TknU8   end2EndMethInd;  /* end to end method indcatr     */
    TknU8   intInd;          /* interworking indicator        */
    TKnU8   segInd;          /* simple segmentation indicator */
    TknU8   end2EndInfoInd;  /* end to end info indicator     */
    TknU8   isdnUsrPrtInd;   /* ISDN User Part indicator      */
    TknU8   holdInd;         /* holding indicator             */
    TknU8   isdnAccInd;      /* ISDN access indicator         */
    TknU8   echoCtrlDevInd;  /* echo control device ind.      */
    TknU8   sccpMethInd;     /* SCCP method indicator         */
    TknU8   spare;           /* spare bits                    */
} SiBckCalInd;
```

The fields in the SiBckCallInd structure are encoded as follows:

Field	Value
chrgInd	Charge indicator. Defined values: 0x00 = CHRG_NOIND No indication 0x01 = CHRG_NOCHRG No charge 0x02 = CHRG_CHRG Charge
cadPtyStatInd	Called party's status indicator. Defined values: 0x00 = CADSTAT_NOIND No indication 0x01 = CADSTAT_SUBFREE Subscriber free 0x02 = CADSTAT_CONNFREE Connect when free (national use) 0x03 = CADSTAT_DELAY Excessive delay; spare in BICC
cadPtyCatInd	Called party's category indicator. Defined values: 0x00 = CADCAT_NOIND No indication 0x01 = CADCAT_ORDSUBS Ordinary subscriber 0x02 = CADCAT_PAYPHONE Payphone
end2EndMethInd	End-to-end method indicator. Defined values for variants other than BICC: 0x00 = E2EMTH_NOMETH No end-to-end method available 0x01 = E2EMTH_PASSALNG Pass-along method available (national use) 0x02 = E2EMTH_SCCPMTH SCCP method available 0x03 = E2EMTH_BOTH Pass-along and SCCP methods available (national use) Defined values for BICC: 0x00 = E2EMTH_NOMETH No end-to-end method available ox01 = E2EMTH_RSVRD1 Reserved 0x02 = E2EMTH_RSVRD2 Reserved 0x03 = E2EMTH_RSVRD3 Reserved
intInd	Interworking indicator. Defined values: 0x00 = INTIND_NOINTW No end-to-end information available 0x01 = INTIND_INTW End-to-end information available

Field	Value
segInd	Segmentation indicator. Defined values: 0x00 = SEGIND_NOIND No indication 0x01 = SEGIND_INFO Additional information will be sent
end2EndInfoInd	End-to-end information indicator (national use). Defined values for variants other than BICC: 0x00 = E2EINF_NOINFO No end-to-end information available 0x01 = E2EINF_INFO End-to-end information available Defined values for BICC: 0x00 = E2EINF_NOINFO No end-to-end information available 0x01 = E2EINF_RSVRD Reserved
isdnUsrPrtInd	ISDN user part indicator. Defined values for variants other than BICC: 0x00 = ISUP_NOTUSED ISDN user part not used all the way 0x01 = ISUP_USED ISDN user part used all the way Defined values for BICC: 0x00 = BICC_NOTUSED BICC not used all the way 0x01 = BICC_USED BICC used all the way
holdInd	Holding indicator (national use). Defined values: 0x00 = HOLD_NOTREQD Holding requested 0x01 = HOLD_REQD Holding not requested
isdnAccInd	ISDN access indicator. Defined values: 0x00 = ISDNACC_NONISDN Terminating access non-ISDN 0x01 = ISDNACC_ISDN Terminating access ISDN
echoCtrlDevInd	Echo control device indicator. Defined values: 0x00 = ECHOCDEV_NOTINCL Incoming echo control device not included 0x01 = ECHOCDEV_INCL Incoming echo control device included
sccpMethInd	SCCP method indicator. Defined values for non-BICC variants: 0x00 = SCCPMTH_NOIND No indication 0x01 = SCCPMTH_CONLESS Connectionless model available (national use) 0x02 = SCCPMTH_CONORNTD Connection oriented method available 0x03 = SCCPMTH_BOTH Connectionless and connection oriented methods available (national use) Defined values for BICC: 0x00 = SCCPMTH_NOIND No indication 0x01 = SCCPMTH_RSVRD1 Reserved 0x02 = SCCPMTH_RSVRD2 Reserved 0x03 = SCCPMTH_RSVRD3 Reserved

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	ANSI 92	ANSI 95	BICC	ETSI V2	ETSI V3
chrgInd	*	*	*	*	*	*
cadPtyStatInd	*	*	*	*	*	*
cadPtyCatInd	*	*	*	*	*	*
end2EndMethInd	*	*	*	*	*	*
intInd	*	*	*	*	*	*
segInd		*	*			
end2EndInfoInd	*			*	*	*
isdnUsrPrtInd	*	*	*	*	*	*
holdInd	*	*	*	*	*	*
isdnAccInd	*	*	*	*	*	*
echoCtrlDevInd		*	*	*	*	*
sccpMethInd		*	*	*	*	*
spare	N-P					

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White	ITU 97	Q.767
chrgInd	*	*	*	*
cadPtyStatInd	*	*	*	*
cadPtyCatInd	*	*	*	*
end2EndMethInd	*	*	*	*
intInd	*	*	*	*
segInd				
end2EndInfoInd	*	*	*	*
isdnUsrPrtInd	*	*	*	*
holdInd	*	*	*	*
isdnAccInd	*	*	*	*
echoCtrlDevInd	*	*	*	*
sccpMethInd	*	*	*	*
spare				

Backward GVNS

Associated variants: BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Describes the global virtual network service in the backward direction.

```
typedef struct_bckGVNS
{
    ElmtHdr eh;             /* element header              */
    TknU8   termAccInd;     /* Terminating access indicator */
    TknU8   spare1;         /* bits c-g                    */
}SiBckGVNS;
```

The termAccInd field is encoded to one of the following values:

0	TERM_ACC_NO_INFO	No information.
1	TERM_ACC_DEDICATED	Dedicated terminating access.
2	TERM_ACC_SWITCHED	Switched terminating access.
3	TERM_ACC_SPARE	Spare.

Tokens

Token	BICC	ETSI V2	ETSI V3	ITU White	ITU 97
termAccInd	*	*	*	*	*
spare1	C-G	C-G	C-G	C-G	C-G

Business group

Associated variants: ANSI 92, ANSI 95

Identifies the properties of a group of subscriber lines that belong to a common subscriber, such as a Centrex group.

```
typedef struct _businessGrp    /* Business Group                */
{
    ElmtHdr eh;                /* element header                */
    TknU8   partySel;          /* party selector                */
    TknU8   linePrivInfInd;    /* line privileges info ident.   */
    TknU8   BGIDident;         /* BGID identifier               */
    TknU8   attendStat;        /* attendant status              */
    TknU32  busiGrpIdent;      /* business group ident.         */
    TknU16  subGrpIdent;       /* sub-group identifier          */
    TknU8   linePriv;          /* line privileges               */
} SiBusinessGrp;
```

The fields in the SiBusinessGrp structure are encoded as follows:

Field	Value
partySel	0x00 = PRTY_NOIND No indication 0x01 = PRTY_CGPTYNUM Calling party number 0x02 = PRTY_CDPTYNUM Called party number 0x03 = PRTY_CONNDPTYNUM Connected party number 0x04 = PRTY_REDIRGNUM Redirecting number 0x05 = PRTY_ORIGCALLNUM Original called number
linePrivInfInd	Line privileges information identifier. Defined values: 0x00 = PRIV_FIXED Fixed line privileges 0x01 = PRIV_CUSTDEF Customer-defined line privileges
BGIDident	Business group identifier (BGID) type. Defined values: 0x00 = BGID_MULTILOC Multi location business group identifier 0x01 = BGID_INTERNET Internetworking with private networks identifier
attendStat	Attendant status. Defined values: 0x00 = ATTEN_NOIND No indication 0x01 = ATTEN_ATTENDLINE Attendant line
busiGrpIdent	Business group identifier (only the least significant 24 bits are used). Defined values: 0 = no indication 1 = public network All other values are network dependent.
subGrpIdent	Subgroup identifier (16 bits). Value zero is no subgroups. All other values represent a subgroup number.
linePriv	Line privileges. Defined values: 0x00 = LP_RESTRICT Unrestricted 0x01 = LP_SEMIRESTRICT Semi-restricted 0x02 = LP_FULLRESTRICT Fully restricted 0x03 = LP_FULLRESTRICT_INSWTCH Fully restricted, intra-switch 0x04 = LP_DENIED Denied

Tokens

Token	ANSI 92	ANSI 95
partySel	*	*
linePrivInfInd	*	*
BGIDident	*	*
attendStat	*	*
busiGrpIdent	*	*
subGrpIdent	*	*
linePriv	*	*

Call diversion information

Associated variants: BICC, ITU White

Contains information sent in the backward direction to notify the originating exchange of the redirecting reason and the notification subscription options of the redirecting party.

```
typedef struct _cllDivr /* call Diversion information  */
{
    ElmtHdr eh;         /* element header              */
    TknU8   notSuscr;   /* Notification subscription   */
    TknU8   redirRsn;   /* redirection reason          */
    TknU8   spare;      /* spare bits                  */
} SiCllDiverInfo;
```

The fields in the SiCllDiverInfo structure are encoded as follows:

Field	Description
notSuscr	Notification subscription. Defined values: 0x00 = PRES_UNKNOWN Unknown 0x01 = PRES_NOTALLOW Presentation not allowed 0x02 = PRES_ALLOWWREDNUM Presentation allowed with redirection number 0x03 = PRES_ALLOW Presentation allowed without redirection number
redirRsn	Redirecting reason. Defined values: 0x00 = REAS_UNKNWN Unknown 0x01 = REAS_USRBUSY User busy 0x02 = REAS_NOREPLY No reply 0x03 = REAS_UNCOND Unconditional 0x04 = REAS_DFLCDURALRT Deflection during alerting 0x05 = REAS_DFLCIMMDRSP Deflection immediate response 0x06 = REAS_MBLSUBNOTRCHBL Mobile subscriber not reachable

Tokens

Token	BICC	ITU White
notSuscr	*	*
redirRsn	*	*
spare	H	H

Call diversion treatment indicators

Associated variants: BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Identifies the available options when allowing calls to be diverted.

```
typedef struct_callDivTrtmnt     /* Call diversion treatment Ind  */
{
    ElmtHdr eh;                  /* element header                */
    TknU8   callToBeDiv;         /* call to be diverted indicator */
    TknU8   spare1;              /* bits C-G                      */
}SiCallDivTrtmnt;
```

The callToBeDiv field is encoded to one of the following values:

0	CALL_DIV_NO_INDICATION	No indication.
1	CALL_DIV_ALLOWED	Call diversion allowed.
2	CALL_DIV_NOT_ALLOWED	Call diversion not allowed.
3	CALL_DIV_SPARE	Spare.

Tokens

Token	BICC	ETSI V2	ETSI V3	ITU White	ITU 97
callToBeDiv	*	*	*	*	*
spare1	C-G	C-G	C-G	C-G	C-G

Call modification indicators

Associated variants: ANSI 88, ITU Blue

Supports in-call modification in ITU-T (CCITT) 1988 networks. In-call modification is not supported in ANSI networks and has been removed from the ITU-T 1992 standards.

```
typedef struct _calModInd  /* Call Modification Indicators */
{
    ElmtHdr eh;            /* element header               */
    TknU8   modInd;        /* call modification indicators */
    TknU8   spare;         /* spare bits                   */
} SiCalModInd;
```

The modInd field is encoded to one of the following values:

0x01	MOD_SERV1
	MOD_SERV2

Tokens

Token	ANSI 88	ITU Blue
modInd	*	*
spare	C-H	C-H

Call offering treatment indicators

Associated variants: BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Identifies the options for allowing calls to be offered.

```
typedef struct_callOffTrtmnt  /* Call offering treatment Ind  */
{
    ElmtHdr eh;               /* element header                */
    TknU8   callToBeOff;      /* call to be offered indicator  */
    TknU8   spare1;           /* bits C`-G                      */
}SiCallOffTrtmnt;
```

The callToBeOff field is encoded to one of the following values:

0X00	CALL_OFF_NO_INDICATION	No indication.
0X01	CALL_OFF_NOT_ALLOWED	Call offering not allowed.
0X10	CALL_OFF_ALLOWED	Call offering allowed.
0X11	CALL_OFF_SPARE	Spare.

Tokens

Token	BICC	ETSI V2	ETSI V3	ITU White	ITU 97
callToBeOff	*	*	*	*	*
spare1	C-G	C-G	C-G	C-G	C-G

Call reference

Associated variants: All except Q.767

Identifies the reference number assigned to a call for use in subsequent messages related to that call. This number has meaning only to the exchange that assigns it. This information element is relevant to BICC only in MTP3- and MTP3b-based signaling networks.

Note: For ANSI networks, use the callRefA structure that supports 24-bit point codes. For ITU-T and BICC, and ETSI networks, use the callRef structure that supports 14-bit point codes. This parameter is relevant to BICC only in MTP 3- and MTP 3b-based signaling networks.

```
typedef struct _callRef      /* Call Reference */
{
    ElmtHdr eh;              /* element header */
    TknU32  callId;          /* call identity  */
    TknU16  pntCde;          /* point code     */
} SiCallRef;

typedef struct _callRefA     /* Call Reference */
{
    ElmtHdr eh;              /* element header */
    TknU32  callId;          /* call identity  */
    TknU32  pntCde;          /* point code     */
} SiCallRefA;
```

The fields in the SiCallRef and SiCallRefA structures are encoded as follows:

Field	Description
callID	Call identity encoded as a 32-bit quantity of which the least significant 24 bits are used.
pntCde	Signaling point code for the call encoded as follows: • In ANSI, a 32-bit quantity of which the least significant 24 bits are used. • In BICC, ETSI, and ITU-T, a 16-bit quantity of which the least significant 14 bits are used. For example, an ANSI point code represented by the decimal string 1.4.7 is encoded as the hexadecimal number 0x00010407.

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	ANSI 92	ANSI 95	BICC	ETSI V2	ETSI V3
callId	*	*	*	*	*	*
pntCde	*	*	*	*	*	*

Tokens for the ITU variants

Token	ITU Blue	ITU White	ITU 97
callId	*	*	*
pntCde	*	*	*

Call transfer number

Describes the number to which a call is transferred.

```
typedef struct_callXferNum    /* Call transfer number                  */
{
    ElmtHdr eh;               /* element header                        */
    TknU8   natAddrInd;       /* nature of address indicator           */
    TknU8   oddEven;          /* odd or even                           */
    TknU8   scrnInd;          /* screen indicator                      */
    TknU8   presRest;         /* Address presentation restricted ind. */
    TknU8   numPlan;          /* numbering plan                        */
    TknU8   spare1;           /* spare bits                            */
    TknU8   addrSig;          /* Address signal                        */
}SiCallXferNum;
```

Refer to *Called party number* on page 201 and *Calling party number* on page 205 for field values.

Call transfer reference

Associated variants: BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Provides a binary representation of the integer assigned unambiguously to the particular ECT supplementary service invocation.

```
typedef struct_callXferRef     /* Call transfer reference    */
{
    ElmtHdr eh;                /* element header             */
    TknU8   cllXferRef;;       /* call transfer value 0-255  */
}SiCallXferRef;
```

Tokens

Token	BICC	ETSI V2	ETSI V2	ITU White	
cllXferRef	*	*	*	*	*

Called IN number

Provides the called IN number format.

```
typedef struct_cdINNum         /* Called IN number                       */
{
    ElmtHdr eh;                /* element header                         */
    TknU8   natAddrInd;        /* nature of address indicator            */
    TknU8   oddEven;           /* odd or even                            */
    TknU8   spare1;            /* spare bits                             */
    TknU8   scrnInd;           /* screen indicator                       */
    TknU8   presRest;          /* Address presentation restricted ind.  */
    TknU8   numPlan;           /* numbering plan                         */
    TknU8   spare2;            /* spare bits                             */
    TknU8   addrSig;           /* Address signal                         */
}SiCdINNum;
```

Refer to *Called party number* on page 201 and *Calling party number* on page 205 for field values.

Called party number

Associated variants: All

Contains the information necessary to identify the called party.

```
typedef struct _cdPtyNum    /* Called Party Number               */
{
    ElmtHdr eh;             /* element header                    */
    TknU8   natAddrInd;     /* nature of addr indicator          */
    TknU8   oddEven;        /* odd or even                       */
    TknU8   spare;          /* spare bits                        */
    TknU8   numPlan;        /* numbering plan                    */
    TknU8   reserved;       /* reserved bits                     */
    TknU8   innInd;         /* internal network number *indic.   */
    TknStr  addrSig;        /* Address Signal                    */
} SiCdPtyNum;
```

The fields in the SiCdPtyNum structure are encoded as follows:

Field	Values
natAddrInd	Nature of address indicator. Defined values for all variants: 0x01 = SUBSNUM Subscriber number 0x03 = NATNUM Nationally significant number 0x04 = INTNATNUM International number Additional values for ANSI: 0x72 = NATNUMOPREQ National number, operator requested 0x73 = INTNATNUMOPREQ International number, operator requested 0x74 = NONUMPRESOPREQ No number present, operator requested 0x75 = NONUMPRESCUTTHRU No number present, cut-through call to carrier 0x76 = NINEFIVEOH 950+ service 0x77 = TSTLINETSTCODE Test line test code
oddEven	Specifies whether the number of digits in the addrSig field is even or odd. If even, the last octet contains two digits. If odd, the last octet contains only one digit and the most significant four bits not used. Defined values: 1 = NMB_ODD
spare	Spare bits.
numPlan	Numbering plan. Defined values for all supported variants except BICC: 0x00 = NP_UNK Unknown 0x01 = NP_ISDN ISDN/telephony according to ITU-T E.164 0x02 = NP_TEL Telephony numbering according to E.163; spare in ITU Blue, ITU White, and ITU 97 0x03 = NP_DATA Data numbering according to X.121 0x04 = NP_TELEX Telex number according to F.69 0x08 = NP_NATIONAL National standard numbering 0x09 = NP_PRIVATE Private numbering plan 0x0f = NP_EXT Reserved for extension Defined values for BICC: 0x00 = NP_UNK Unknown 0x02 = NP_TEL Spare 0x03 = NP_DATA Data numbering according to ITU-T X.121 0x04 = NP_TELEX Telex number according to ITU_T F.69 0x05 = NP_PRIVATE Private numbering plan.
innInd	Internal network indicator. Defined values: 0x00 = INN_ALLOW Routing to an internal network number allowed 0x01 = INN_NOTALLOW Routing to an internal network number not allowed
addrSig	Actual address digits, encoded as shown in the following tables.

Actual address digits, encoded as follows:

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

Where each address digit is encoded with the following bit pattern:

Bit pattern	Digit/signal
0000	0
0001	1
0010	2
	3
	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

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	ANSI 92		BICC	ETSI V2	ETSI V3
natAddrInd	*	*	*	*	*	*
oddEven	*	*	*	*	*	*
spare	2(8)	2(8)	2(8)	*		
numPlan	*	*	*	*	*	*
reserved	2(1-4)	2(1-4)	2(1-4)		2(1-4)	2(1-4)
innInd				*	*	*
addrSig	*	*	*	*	*	*

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White		Q.767
natAddrInd	*	*	*	*
oddEven	*	*	*	*
spare				
numPlan	*	*	*	*
reserved	2(1-4)	2(1-4)	2(1-4)	2(1-4)
innInd	*	*	*	*
addrSig	*	*	*	*

Calling party category

Associated variants: All

Contains information sent in the forward direction to indicate the type of the originating party and, possibly for operator-assisted calls, the desired service language.

```
typedef struct _cgPtyCat     /* Calling Party Category     */
{
    ElmtHdr eh;              /* element header             */
    TknU8   cgPtyCat;        /* calling party category     */
} SiCgPtyCat;
```

The cgPtyCat field is encoded to one of the following values:

0x00	CAT_UNKNOWN	Unknown (default).
0x01	CAT_OPLANGFR	French language operator.
0x02	CAT_OPLANGENG	
0x03		German language operator.
0x04	CAT_OPLANGRUS	Russian language operator.
0x05	CAT_OPLANGSP	Spanish language operator.
0x06	CAT_ADMIN1	
0x07	CAT_ADMIN2	Available to administrators.
0x08	CAT_ADMIN3	Available to administrators.
0x0A	CAT_ORD	Ordinary subscriber.
0x0B	CAT_PRIOR	Priority subscriber.
0x0C	CAT_DATA	Data call.
0x0D	CAT_TEST	Test call.
0x0F	CAT_PAYPHONE	Pay phone.

Tokens for the ASNI, BICC, and ETSI variants

Token	ANSI 88	ANSI 92	ANSI 95	BICC	ETSI V2	ETSI V3
cgPtyCat	*	*	*	*	*	*

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White	ITU 97	Q.767
cgPtyCat	*	*	*	*

Calling party number

Associated variants: All

Provides the format of the party placing a call (the caller).

```
 typedef struct _cgPtyNum  /* Calling Party Number              */
{
  ElmtHdr  eh;             /* element header                    */
  TknU8    natAddrInd;     /* nature of address indicator       */
  TknU8    oddEven;        /* odd or even                       */
  TknU8    scrnInd;        /* screen indicator                  */
  TknU8    presRest;       /* Addr presentation restricted ind. */
  TknU8    numPlan;        /* numbering plan                    */
  TknU8    niInd;          /* number incomplete indicator       */
  TknU8    spare;          /* spare bits                        */
  TknStr   addrSig;        /* Address Signal                    */
} SiCgPtyNum;
```

The fields in the SiCgPtyNum structure are encoded as follows:

Field	Values
natAddrInd	0x01 = SUBSNUM Subscriber number 0x03 = NATNUM Nationally significant number 0x04 = INTNATNUM International number Additional values for ANSI: 0x71 = SUBSNUMOPREQ Subscriber number operator requested 0x72 = NATNUMOPREQ National number operator requested 0x73 = INTNATNUMOPREQ International number operator requested 0x74 = NONUMPRESOPREQ No number present operator requested 0x75 = NONUMPRESCUTTHRU No number present cut-through call to carrier 0x77 = TSTLINETSTCODE Test line test code
oddEven	Specifies whether the number of digits in the addrSig field is even or odd. If even, the last octet contains two digits. If odd, the last octet contains only one digit and the most significant four bits not used. Defined values: 0 = NMB_EVEN 1 = NMB_ODD
scrnInd	0x00 = USRPROVNOTVER User provided, not verified 0x01 = USRPROV User provided, verified passed 0x02 = USRPROVVERFAIL User provided, verified failed 0x03 = NETPROV Network provided
	Address presentation restricted indicator. Defined values: 0x00 = PRESALLOW Presentation allowed 0x01 = PRESREST Presentation restricted

Field	**Values**
numPlan	Numbering plan. Defined values for all supported variants except BICC: 0x00 = NP_UNK Unknown 0x01 = NP_ISDN ISDN/telephony according to E.164/E.163 0x02 = NP_TEL Telephony numbering according to E.163; spare in ITU Blue, ITU White, and ITU 97 0x03 = NP_DATA Data numbering according to X.121 0x04 = NP_TELEX Telex number according to F.69 0x08 = NP_NATIONAL National standard numbering 0x09 = NP_PRIVATE Private numbering plan 0x0f = NP_EXT Reserved for extension Defined values for BICC: 0x00 = NP_UNK Unknown 0x01 = NP_ISDN ISDN/telephony according to ITU-T E.164/E.163 0x02 = NP_TEL Spare 0x03 = NP_DATA Data numbering according to ITU-T X.121 0x04 = NP_TELEX Telex number according to ITU_T F.69 0x05 = NP_PRIVATE Private numbering plan. 0x06 = NP_NATIONAL Reserved for national use
niInd	Number incomplete indicator. Defined values: 0x00 = NBMCMLTE Number complete
addrSig	Actual address digits, encoded as shown in the following tables.

For the addrSig field, the actual address digits are encoded as follows:

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

where each digit is encoded with the following bit pattern:

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

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	ANSI 92	ANSI 95	BICC	ETSI V2	ETSI V3
natAddrInd	*	*	*	*	*	*
oddEven	*	*	*	*	*	*
scrnInd	*	*	*	*	*	*
presRest	*	*	*	*	*	*
numPlan	*	*	*	*	*	*
niInd				*	*	*
spare	2(8)	2(8)	2(8)			
addrSig	*	*	*	*	*	*

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White	ITU 97	Q.767
natAddrInd	*	*	*	*
oddEven	*	*	*	*
scrnInd	*	*	*	*
presRest	*	*	*	*
numPlan	*	*	*	*
niInd	*	*	*	*
spare				
addrSig	*	*	*	*

Carrier ID

Associated variants: ANSI 92, ANSI 95

Specifies the carrier used for a connection in ANSI networks.

```
typedef struct _carrierId   /* Carrier ID               */
{
    ElmtHdr eh;             /* element header           */
    TknU8   netIdPln;       /* network id plan          */
    TknU8   typNetId;       /* Network id type          */
    TknU8   spare;          /* spare bits               */
    TknU8   CIDigit1;       /* Network Identity Digit 1 */
    TknU8   CIDigit2;       /* Network Identity Digit 2 */
    TknU8   CIDigit3;       /* Network Identity Digit 3 */
    TknU8   CIDigit4;       /* Network Identity Digit 4 */
} SiCarrierId;
```

Fields in the SiCarrierId structure are encoded as follows:

Field	Value
netIdPlnl	Network ID plan. Defined values: 0x00 = NI_UNKNWN 0x01 = NI_3DIGCIC
typNetId	Defined values: 0x00 = TNI_CCITT
spare	Spare bits.
CIDigit1 CIDigit2 CIDigit3 CIDigit4	Carrier identification digits, encoded as shown in the following tables.

For the CIDigit***n*** field, the address digits are encoded as follows:

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

where each digit is encoded with the following bit pattern:

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

Tokens

Token	ANSI 92	ANSI 95
netIdPln1	*	*
typNetId2	*	*
spare	1(8)	1(8)
CIDigit1	*	*
CIDigit2	*	*
CIDigit3	*	*
CIDigit4	*	*

Carrier selection information

Associated variants: ANSI 92 and ANSI 95

Specifies how the carrier for a connection is selected.

```
typedef struct _carrierSelInf /* Carrier Selection Info */
{
    ElmtHdr eh;               /* element header         */
    TknU8   carrierSelInf;    /* carrier selection info */
} SiCarrierSelInf;
```

The SiCarrierSelInf field is encoded as follows:

0x00	CARSEL_NOIND	No indication.
0x01	CARSEL_PRESUB_NOINPUT	Selected carrier identification is pre-subscribed and not input by the calling party.
0x02	CARSEL_PRESUB_INPUT	Selected carrier identification is pre-subscribed and input by the calling party.
0x03	CARSEL_PRESUB_INPUTUNDET	Selected carrier identification is pre-subscribed and input by the calling party is undetermined.
0x04	CARSEL_NOTPRESUB_INPUT	Selected carrier identification is not pre-subscribed, and input by the calling party.

Tokens

Token	ANSI 92	ANSI 95
carrierSelInf	*	*

Cause indicator

Associated variants: All

Identifies the cause of a failure, disconnect, or rejected message.

```
typedef struct _siCauseDgn   /* Cause Indicator */
{
    ElmtHdr eh;              /* element header  */
    TknU8   location;        /* location        */
    TknU8   spare;           /* spare bits      */
    TknU8   cdeStand;        /* coding standard */
    TknU8   recommend;       /* recommendation  */
    TknU8   causeVal;        /* cause value     */
    TknStr  dgnVal;          /* diagnostics     */
} SiCauseDgn;
```

The fields in the SiCauseDgn structure are encoded as follows:

Field	Value
location	Location. Defined values: 0x00 = ILOC_USER User 0x01 = ILOC_PRIVNETLU Private network serving local user 0x02 = ILOC_PUBNETLU Public network serving local user 0x03 = ILOC_TRANNET Transit network 0x04 = ILOC_PRIVNETRU Private network serving the remote user 0x05 = ILOC_PUBNETRU Public network serving the remote user 0x07 = ILOC_INTNET International network 0x0a = ILOC_NETINTER Network beyond inter-networking point 0x0f = ILOC_NOINFOAV No information concerning origin location (Not in BICC)
cdeStand	Coding standard. Defined values: 0x00 = CSTD_CCITT CCITT standards 0x01 = CSTD_INT Reserved for other international standards 0x02 = CSTD_NAT National standard 0x03 = CSTD_SPECLOC Standard specific to identified location 0x03 = CSTD_NET Standard specific network (BICC)
recommend	Recommendation. Defined values: 0x00 = REC_Q763 CCITT Recommendation Q.763 0x03 = REC_X21 CCITT Recommendation X.21 0x04 = REC_X25 CCITT Recommendation X.25 0x05 = REC_Q1000 CCITT Recommendation Q.1000
causeVal	Cause value. For more information, refer to *Defined values for causeVal by class* on page 213
dgnVal	Structure of the diagnostic field depends on the cause value (causeVal) field. The ISUP layer does not interpret the contents but passes the value through as a transparent string of octets. The application must encode or interpret the string of octets as specified for the associated cause value in the relevant variant recommendations.

Defined values for causeVal by class

The following table lists the defined values for the causeVal field by class:

Class	Description	Values
000 and 001	Normal events	1 = CCUNALLOC Unassigned number 2 = CCNORTTOTSFNET No route to transit net 3 = CCNORTTODEST No route to destination 4 = CCSENDSPCLTONE Send special info tone 5 = CCMISDIALDTRNK Misdialed trunk prefix 16 = CCCALLCLR Normal call clearing 17 = CCUSRBSY User busy 18 = CCNOUSRRSP No user response 19 = CCNOANSWR No answer (user alerted) 21 = CCCALLRJT Call rejected 22 = CCNMBRCHG Number changed 27 = CCDESTOUTORD Destination out of order 28 = CCADDRINCOMP Address incomplete 29 = CCFACREJ Facility rejected 31 = CCNORMUNSPEC Normal unspecified
010	Resource unavailable	34 = CCNOCIRCUIT No circuit/channel available 38 = CCNETAOL Network out of order 41 = CCTMPFAIL Temporary failure 42 = CCSWTCHCONG Switch equipment congestion 44 = CCREQUNAVAIL Requested circuit or channel unavailable 47 = CCRESCUNAVAIL Resource unavailable or unspecified 43 = CCUSRINFDISCARD User information discarded 47 = CCPREEMPT Preemption
011	Service or option not available	50 = CCFACNOTSUB Facility not subscribed 55 = CCINCBARRDCUG Incoming calls barred within CUG 57 = CCNOTAUTHBCAP Bearer capability not authorized 63 = CCSERVUNAVAIL Service or option unavailable
100	Service or option not implemented	65 = CCBCAPNOTIMP Bearer capability not implemented 69 = CCFACNOTIMP Facility not implemented 70 = CCRESTDIG Only restricted digital bearer capability is available 79 = CCSERVNOTIMP Service or option not implemented
101	Invalid message	87 = CCCUNOTMEMBR Called user not member of CUG 88 = CCINCOMPDEST Incompatible destination 91 = CCINVTRNSTNET Invalid transit network selection 95 = CCINVMSG Invalid message unspecified ANSI only 81 = CCINVALCALLREF Invalid call reference value

Class	Description	Values
110	Protocol error	96 = CCINFOELMSSG Mandatory information element is missing 97 = CCNOMSGTYP Message type is non-existent or not implemented 99 = CCNOPARAMDISC Parameter non-existent or not implemented - discard 102 = CCTMRRECOV Timeout recovery 103 = CCNOPARAMPASS Parameter non-existent or not implemented pass along 111 = CCPROTERR Protocol error, unspecified ANSI only 100 = CCINVALPARAMCONT Invalid parameter contents
111	Internetworking	127 = CCINTRWRK Interworking unspecified

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	ANSI 92	ANSI 95	BICC	ETSI V2	ETSI V3
location	*	*	*	*	*	*
spare	1(5)	1(5)	1(5)	1(5)	1(5)	1(5)
cdeStand	*	*	*	*	*	*
recommend						
causeVal	*	*	*	*	*	*
dgnVal		*	*	*	*	*

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White	ITU 97	Q.767
location	*	*	*	*
spare	1(5)	1(5)	1(5)	1(5)
cdeStand	*	*	*	*
recommend	*			
causeVal	*	*	*	*
dgnVal	*	*	*	

CCBS

Associated variant: ETSI V2

Provides the completion of call to busy subscriber (CCBS) format. This information element uses the SiCCSS structure.

```
typedef struct _ccSS            /* CCSS (CCBS in ETSI V2)  */
ElmtHdr   ehement header        /* element header          */
    TknU8     ccssCallInd;      /* CCSS call indicator     */
    TknU8     spare1;           /* bits B-H                */
} SiCCSS;
```

The ccssCallInd field is encoded to one of the following values:

0	CCSS_NO_INDICATION
1	CCSS_CALL

Tokens

Token	ETSI V2
ccssCallInd	*
spare1	B-H

CCNR possible indicator

Associated variants: BICC, ETSI V3

Provides the completion of calls on no reply (CCNR) format.

```
typedef struct _ccnr      /* CCNR possible indicator  */
{
    ElmtHdr eh;           /* element header           */
    TknU8   ccnrPossInd;  /* CCNR possible indicator  */
    TknU8   spare1;       /* bits B-H                 */
}SiCCNR;
```

The ccnrPossInd field is encoded as follows:

0	CCNR_NOT_POSSIBLE
1	CCNR_POSSIBLE

Tokens

Token	BICC	ETSI V3
ccnrPossInd	*	*
spare1	B-H	B-H

CCSS

Associated variants: BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Provides the call completion on service setup (CCSS) format.

```
typedef struct _ccSS            /* CCSS (CCBS in ETSI V2)  */
    ElmtHdr   ehement header    /* element header          */
    TknU8     ccssCallInd;      /* CCSS call indicator     */
    TknU8     spare1;           /* bits B-H                */
} SiCCSS;
```

The ccssCallInd field is encoded to one of the following values:

0	CCSS_NO_INDICATION
1	CCSS_CALL

Tokens

Token	BICC	ETSI V2	ETSI V3	ITU White	ITU 97
ccssCallInd	*	*	*	*	*
spare1	B-H	B-H	B-H	B-H	B-H

Charge number

Associated variants: ANSI 88, ANSI 92, ANSI 95

Passes a charge number between signaling points in ANSI networks.

```
typedef struct _chargeNum    /* Charge Number                */
{
    ElmtHdr eh;              /* element header               */
    TknU8   natAddrInd;      /* nature of address indicator  */
    TknU8   oddEven;         /* odd or even                  */
    TknU8   reserved;        /* reserved bits                */
    TknU8   numPlan;         /* numbering plan               */
    TknU8   spare;           /* spare bits                   */
    TknStr  addrSig;         /* Address Signal               */
} SiChargeNum;
```

Refer to *Called party number* on page 201 for field values.

Tokens

Token	ANSI 88	ANSI 92	ANSI 95
natAddrInd	*	*	*
oddEven	*	*	*
reserved	2(1-4)	2(1-4)	2(1-4)
numPlan	*	*	*
spare	2(8)	2(8)	2(8)
addrSig	*	*	*

Charged party identification

Associated variant: ITU 97

Information that identifies the charged party. Charged party identification is an extended element. Refer to *Extended element* on page 185 for more information.

Circuit assignment map for ANSI

Associated variant: ANSI 95

Provides the circuit assignment map for ANSI. Used in the setup and management of NxDS0 connections.

```
typedef struct cirAssignMap   /* Circuit Assignment Map           */
{
    ElmtHdr eh;               /* element header                   */
    TknU8   mapFormat;        /* map type                         */
    TknU8   spare;            /* spare bits                       */
    TknU8   map;              /* assignment map                   */
} SiCirAssignMap;
```

The fields in the SiCirAssignMap structure are encoded as follows:

Field	Value
mapFormat	Map format. Only defined value is: 0x01 = MAP_DSI DSI map format
map	Map type.

Tokens

Token	ANSI 95
mapFormat	*
spare	*
map	*

Circuit assignment map for ETSI and ITU

Associated variants: ETSI V2, ETSI V3, ITU White, ITU 97

Provides the circuit assignment map for ETSI, and ITU.

```
typedef struct _cirAssMap
{
    ElmtHdr eh;         /* element header */
    TknU8   mapType;    /* map type */
    TknU8   spare1;     /* bits G-H in first byte */
    TknU8   map1;       /* bits A-H - a bit map of used circuits */
    TknU8   map2;       /* bits A-H */
    TknU8   map3;       /* bits A-H  */
    TknU8   map4;       /* bits A-G   */
    TknU8   spare2;     /* bits H in last byte  */
}SiCirAssMap;
```

The fields in the SiCirAssMap structure are encoded as follows:

Field	Value
mapType	Map type. Defined values: 1 = MAPTYPE_1544 1544 k/bits digital path map format (64 kbit/s base rate) 2 = MAPTYPE_2048 2048 k/bits digital path map format (64 kbit/s base rate)
map1 map2 map3 map4	Map format. Defined values: 0 = CIRCUIT_NOT_USED 64 kbit/s circuit is not used 1 = CIRCUIT_USED 64 kbit/s circuit is used.

Tokens

Token	ETSI V2	ETSI V3	ITU White	ITU 97
mapType	*	*	*	*
spare1	G-H	G-H	G-H	G-H
map1	A-H	A-H	A-H	A-H
map2	A-H	A-H	A-H	A-H
map3	A-H	A-H	A-H	A-H
map4	A-G	A-G	A-G	A-G
spare2	H	H	H	H

Circuit group characteristics

Associated variants: ANSI 88, ANSI 92, ANSI 95

Contains information sent in response to a circuit validation request from the far exchange.

```
typedef struct _cirGrpCharInd  /* Circuit group characterstic ind.     */
{
    ElmtHdr eh;                /* element header                       */
    TknU8   cirGrpCarInd;      /* circuit grp. carrier ind.            */
    TknU8   dblSzCtrlInd;      /* double seizing control ind.          */
    TknU8   alarmCarInd;       /* alarm carrier indicator              */
    TknU8   contChkReqInd;     /* continuity check requirements ind.   */
} SiCirGrpCharInd;
```

The fields in the SiCirGrpCharInd structure are encoded as follows:

Field	Value
cirGrpCarInd	Circuit group carrier indicator. Defined values: 0x00 = CG_UNKNOWN Unknown 0x01 = CG_ANALOG Analog 0x02 = CG_DIGITAL Digital 0x03 = CG_ANALDIG Digital and analog
dblSzCtrlInd	Double seizing control indicator. Defined values: 0x00 = DS_UNKNOWN Unknown 0x01 = DS_ODDCIC Odd cic control 0x02 = DS_EVENCIC Even cic control Additional value for ANSI 92: 0x03 = DS_ALLCIC All cic control
alarmCarInd	Alarm carrier indicator. Defined values: 0x00 = AC_UNKNOWN Unknown 0x01 = AC_SOFTCARHAND Software carrier handling 0x02 = AC_HARDCARHAND Hardware carrier handling
contChkReqInd	Defined values: 0x00 = CO_UNKNOWN Unknown 0x01 = CO_NONE None 0x02 = CO_STATIS Statistical 0x03 = CO_PERCALL Per call

Tokens

Token	ANSI 88	ANSI 92	ANSI 95
cirGrpCarInd	*	*	*
dblSzCtrlInd	*	*	*
alarmCarInd	*	*	*
contChkReqInd	*	*	*

Circuit/CIC group supervision

Associated variants: All

Instructs the far exchange on the method of circuit blocking (ANSI) or whether blocking is maintenance or hardware failure related (ITU-T).

```
typedef struct _cirGrpSupMTypInd
                             /* Circuit Group Supervision Msg.Type Ind. */
{
    ElmtHdr eh;              /* element header                          */
    TknU8   typeInd;         /* message type ind.                       */
    TknU8   spare;           /* spare bits                              */
} SiCirGrpSupMTypInd;
```

The typeInd field is encoded to one of the following values in all variants except for BICC and ANSI:

0x00	MAINT Maintenance oriented
0X01	HARDFAIL Hardware failure oriented

The type Ind field is encoded to one of the following values in ANSI:

0x00	BLOCK_WO_REL Block without release
0x01	BLOCK_REL Block with immediate release

The typeInd field is coded to one of the following values in BICC:

0x00	MAINT	Maintenance blocked

Tokens for the ANSI, BICC, and ETSI versions

Token	ANSI 88	ANSI 92	ANSI 95	BICC	ETSI V2	ETSI V3
typeInd	*	*	*	*	*	*
spare	C-H	C-H	C-H	C-H	C-H	C-H

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White	ITU 97	Q.767
typeInd	*	*	*	*
spare	C-H	C-H	C-H	C-H

Circuit ID name

Associated variants: ANSI 88, ANSI 92, ANSI 95

Identifies the CLLI name of a trunk to a far exchange.

```
typedef struct _cirIdName   /* Circuit ID Name                   */
{
    ElmtHdr eh;             /* element header                    */
    TknStr  trunkNumClli;   /* trunk number and clli code        */
    TknStr  clliCodeA;      /* clli code A                       */
    TknStr  clliCodeZ;      /* clli code Z                       */
} SiCirIdName;
```

The trunkNumClli parameter is encoded with the ASCII representation of the trunk number (one ASCII digit per octet - 4 octets total) followed by the CLLI name of the associated trunk (one ASCII character per octet).

Tokens

Token	ANSI 88	ANSI 92	ANSI 95
trunkNumClli	*	*	*
clliCodeA	*	*	*
clliCodeZ	*	*	*

Circuit/CIC state indicators

Associated variants: All

Indicates the state of a circuit according to the sending exchange:

```
typedef struct _cirStateInd   /* Circuit State Indicators */
{
    ElmtHdr eh;               /* element header           */
    TknStr  cirSteInd;        /* circuit state indicator. */
} SiCirStateInd;
```

The cirSteInd field is an array of circuit state values for a range of circuits. Each octet is coded in accordance with the relevant variant recommendation.

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	ANSI 92	ANSI 95	BICC	ETSI V2	ESTI V3
cirSteInd	*	*	*	*	*	*

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White	ITU97	Q.767
cirSteInd	*	*	*	*

Circuit validation response

Associated variants: ANSI 88, ANSI 92, ANSI 95

Provides the results of a circuit validation request.

```
typedef struct _cirValRspInd /* Circuit validation response indicator */
{
    ElmtHdr eh;              /* element header                         */
    TknU8   cirValRspInd;    /* user to user info                      */
} SiCirValRspInd;
```

The cirValRspInd field is encoded as follows:

0x00	CV_SUCCESS
0x01	CV_FAILURE

Tokens

Token	ANSI 88	ANSI 92	ANSI 95
cirValRspInd	*	*	*

Closed user group interlock code

Associated variants: ANSI 88, BICC, ETSI V2, ETSI V3, ITU Blue, ITU White, ITU 97, Q.767

Identifies a closed user group within a network. For ANSI 88 networks, use the cugIntCodeA structure. For the other supported networks, use the cugIntCode structure.

```
typedef struct _cugIntCode  /* Closed User Group Interlock Code */
    ElmtHdr eh;             /* element header                   */
    TknU8   dig2;           /* Digit 2                          */
    TknU8   dig1;           /* Digit 1                          */
    TknU8   dig4;           /* Digit 4                          */
    TknU8   dig3;           /* Digit 3                          */
    TknU16  binCde;         /* binary Code                      */
    TknU16  ISDNIdent;      /* ISDN identifier                  */
} SiCugIntCode;
```

The fields in the SiCugIntCode structure are encoded as follows:

Field	Value
dig1 dig2 dig3 dig4	Four digits (binary representation) of the network identity code (0 | 9 + telephone country code or X.121 DNIC).
binCde	16-bit binary code assigned by the network administrator.
ISDNIdent	ISDN network identifier, as per ANSI 1988 recommendations.

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	BICC	ETSI V2	ETSI V3
dig1	*	*	*	*
dig2	*	*	*	*
dig3	*	*	*	*
dig4	*	*	*	*
binCde	*	*	*	*
ISDNIdent	*			

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White	ITU 97	Q.767
dig1	*	*	*	*
dig2	*	*	*	*
dig3	*	*	*	*
dig4	*	*	*	*
binCde	*	*	*	*
ISDNIdent				

Collect call request

Associated variants: BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Provides the collect call request format.

```
typedef struct _collCallReq
{
    ElmtHdr eh;             /* element header            */
    TknU8   collCallReqInd; /* Collect call req indicator */
    TknU8   spare1;         /* bits B-H                  */
}SiCollCallReq;
```

The collCallReqInd field is encoded as follows:

0	COLLECT_NO_INDICATION	No indication.
1	COLLECT_CALL_REQ	Collect call requested.

Tokens

Token	BICC	ETSI V2	ETSI V3	ITU White	ITU 97
collCallReqInd	*	*	*	*	*
spare1	B-H	B-H	B-H	B-H	B-H

Common language location ID

Associated variants: ANSI 88, ANSI 92, ANSI 95

Identifies a signaling point through its CLLI code.

```
typedef struct _clli   /* Common Language Location ID */
{
    ElmtHdr eh;        /* element header              */
    TknStr  clliCode;  /* clli codes                  */
} SiCLLI;
```

The clliCode field is encoded with an ASCII representation of the exchange CLLI code (town, state, building, and so on) as per ANSI recommendations.

Tokens

Token	ANSI 88	ANSI 92	ANSI 95
clliCode	*	*	*

Conference treatment indicator

Associated variants: BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Provides the conference treatment indicator format.

```
typedef struct _confTrtmnt  /* Conference Treatment Ind         */
{
    ElmtHdr eh;             /* element header                   */
    TknU8   confAccInd;     /* conference acceptance indicator */
    TknU8   spare1;         /* bits C-G                         */
}SiConfTrtmnt;
```

The confAccInd field is encoded to one of the following values:

0	CONF_ACC_NO_INDICATION	No indication.
1	CONF_ACC_ACCEPTED	Accept conference request.
2	CONF_ACC_REJECTED	Reject conference request.
3	CONF_ACC_SPARE	Spare.

Tokens

Token	BICC	ETSI V2	ETSI V3	ITU White	
confAccInd	*	*	*	*	*
spare1	C-G	C-G	C-G	C-G	C-G

Connected number

Associated variants: ANSI 88, BICC, ETSI V2, ETSI V3, ITU Blue, ITU White, ITU 97, Q.767

Contains information sent in the backward direction to identify the connected party in ANSI 88 and ITU-T networks.

```
typedef struct _connectedNum     /* Connected number */
{
    ElmtHdr eh;                     /* element header */
    TknU8   natAddr;                /* nature of addresss indicator */
    TknU8   oddEven;                /* odd or even */
    TknU8   scrnInd;                /* screen indicator */
    TknU8   presRest;               /* Address presentation restricted ind. */
    TknU8   numPlan;                /* numbering plan */
    TknU8   spare;
    TknStr  addrSig;                /* Address Signal */
} SiConnectedNum;
```

The fields in the SiConnectedNum structure are encoded as follows:

Field	Value
natAddr	Nature of address indicator. Defined values for all variants: 0x01 = SUBSNUM Subscriber number 0x03 = NATNUM Nationally significant number 0x04 = INTNATNUM International number Additional values for ANSI: 0x71 = SUBSNUMOPREQ Subscriber number operator requested 0x72 = NATNUMOPREQ National number operator requested 0x73 = INTNATNUMOPREQ International number operator requested 0x74 = NONUMPRESOPREQ No number present operator requested 0x75 = NONUMPRESCUTTHRU No number present cut-through call to carrier 0x77 = TSTLINETSTCODE Test line test code 0x76 = NINEFIVEOH 950+ service
oddEven	Specifies whether the number of digits in the addrSig field is even or odd. If even, the last octet contains two digits. If odd, the last octet contains only one digit and the most significant four bits not used. Defined values: 0 = NMB_EVEN 1 = NMB_ODD
scrnInd	Screen indicator. Defined values: 0x00 = USRPROVNOTVER User provided, not verified 0x01 = USRPROV User provided, verified passed 0x02 = USRPROVVERFAIL User provided, verified failed 0x03 = NETPROV Network provided
presRest	Address presentation restricted indicator. Defined values: 0x00 = PRESALLOW Presentation allowed 0x01 = PRESREST Presentation restricted 0x02 = ADDRNOAVAIL Address not available
numPlan	Numbering plan. Defined values for all supported variants except BICC: 0x00 = NP_UNK Unknown 0x01 = NP_ISDN ISDN/telephony according to E.164/E.163 0x02 = NP_TEL Telephony numbering according to E.163; spare in ITU Blue, ITU White, and ITU 97 0x03 = NP_DATA Data numbering according to X.121 0x04 = NP_TELEX Telex number according to F.69 0x08 = NP_NATIONAL National standard numbering 0x09 = NP_PRIVATE Private numbering plan 0x0f = NP_EXT Reserved for extension Defined values for BICC: 0x00 = NP_UNK Unknown 0x01 = NP_ISDN ISDN/telephony according to ITU-T E.164/E.163 0x02 = NP_TEL Spare 0x03 = NP_DATA Data numbering according to ITU-T X.121 0x04 = NP_TELEX Telex number according to ITU_T F.69 0x05 = NP_PRIVATE Private numbering plan. 0x06 = NP_NATIONAL Reserved for national use
addrSig	Actual address digits, encoded as shown in the following tables.

For the addrSig field, the actual address digits are encoded as follows:

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

where each digit is encoded with the following bit pattern:

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

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	BICC	ETSI V2	ETSI V3
natAddr	*	*	*	*
oddEven	*	*	*	*
scrnInd		*	*	*
presRest	*	*	*	*
numPlan	*	*	*	*
spare	2(8)	2(8)	2(8)	2(8)
addrSig	*	*	*	*

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White	ITU 97	Q.767
natAddr	*	*	*	*
oddEven	*	*	*	*
scrnInd	*	*	*	*
presRest	*	*	*	*
spare	2(8)	2(8)	2(8)	2(8)
numPlan	*	*	*	*
addrSig	*	*	*	*

Connection request

Associated variants: All except BICC and Q.767

Contains information sent in the forward direction to request an end-to-end SCCP connection.

```
typedef struct _connReq   /* Connection Request                 */
{
    ElmtHdr eh;           /* element header                     */
    TknU32  locRef;       /* local reference( a 24bit quantity) */
    TknU32  pntCde;       /* point code                         */
    TknU8   protClass;    /* protocol class                     */
    TknU8   credit;       /* credit                             */
} SiConnReq;
```

The fields in the SiConnReq structure are encoded as follows:

Field	Value
locRef	A 24-bit number used by the originating exchange as a reference for this connection.
pntCde	A 32-bit number of which the least significant 24 bits (ANSI) or the least significant 14 bits (ITU-T) are used. For example, an ANSI point code represented by the decimal string 1.4.7 is encoded as hexadecimal number 0x00010407.
protClass	SCCP protocol class (binary encoding) defined in the ANSI or ITU-T recommendations.
credit	

Tokens for the ASNI and ETSI variants

Token	ANSI 88	ANSI 92	ANSI 95	ETSI V2	ETSI V3
locRef	*	*	*	*	*
pntCde	*	*	*	*	*
protClass	*	*	*	*	*
credit	*	*	*	*	*

Tokens for the ITU variants

Token	ITU Blue	ITU White	ITU 97
locRef	*	*	*
pntCde	*	*	*
protClass	*	*	*
credit	*	*	*

Continuity indicators

Associated variants: All

Indicates whether or not a continuity check was successful.

```
typedef struct _contInd  /* Continuity indicators   */
{
    ElmtHdr eh;          /* element header          */
    TknU8   contInd;     /* continuity indicator    */
    TknU8   spare;       /* spare bits              */
} SiContInd;
```

The contInd field is encoded as follows:

Field	Value
contInd	Defined values for all variants except BICC: 0x00 = CONT_CHKFAIL Continuity check failed; reserved in BICC 0x01 = CONT_CHKSUCC Continuity check succeeded; continuity in BICC Defined values for BICC: 0x00 = CONMSG_RSVRD Reserved 0x01 = CONMSG_CONT Continuity

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	ANSI 92	ANSI 95	BICC		ESTI V3
contInd	*	*	*	*	*	*
spare	B-H	B-H	B-H	B-H	B-H	B-H

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White	ITU 97	Q.767
contInd	*	*	*	*
spare	B-H	B-H	B-H	B-H

Correlation ID

Associated variants: BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Provides the correlation ID format.

```
typedef struct_corrID      /* Correlation ID */
{
    ElmtHdr eh;            /* element header */
    TknStr  digits;        /* status         */
}SiCorrelationID;
```

Tokens

Token	BICC	ETSI V2	ETSI V3	ITU White	ITU 97
digits	*	*	*	*	*

Display information

Associated variants: BICC, ITU 97

Allows the communication of a text string. Display information is an extended element. Refer to *Extended element* on page 185 for more information.

Echo control indicators

Associated variants: BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Indicates whether or not a half echo control device is included in the connection.

```
typedef struct _echoControl  /* echo control indicators              */
{
    ElmtHdr eh;              /* element header                       */
    TknU8   outEchoRsp;      /* outgoing echo control device response */
    TknU8   incEchoRsp;      /* incoming echo control device response */
    TknU8   outEchoReq;      /* outgoing echo control device request  */
    TknU8   incEchoReq;      /* incoming echo control device request  */
} SiEchoCtl;
```

The fields in the SiEchoCtl structure are encoded as follows:

Field	Value
outEchoRsp	Outgoing echo control device response. Defined values: 0x00 = ECHCDEV_NOINFOINCL No information 0x01 = ECHCDEV_NOTINCL Device not included 0x02 = ECHCDEV_INCL Device included
incEchoRsp	0x00 = ECHCDEV_NOINFOINCL No information 0x01 = ECHCDEV_NOTINCL Device not included 0x02 = ECHCDEV_INCL Device included
outEchoReq	Outgoing echo control device request. Defined values: 0x00 = ECHCDEV_NOINFOINCL No information 0x01 = ECHCDEV_ACTREQ Device activation request 0x02 = ECHCDEV_DEACTREQ Device deactivation request
incEchoReq	Incoming echo control device request. Defined values: 0x00 = ECHCDEV_NOINFOINCL No information 0x01 = ECHCDEV_ACTREQ Device activation request 0x02 = ECHCDEV_DEACTREQ Device deactivation request

Tokens

Token	BICC	ETSI V2	ETSI V3	ITU White	ITU 97
outEchoRsp	*	*	*	*	*
incEchoRsp	*	*	*	*	*
outEchoReq	*	*	*	*	*
incEchoReq	*	*	*	*	*

Egress service

Associated variants: ANSI 92, ANSI 95

Sends network-specific information regarding a terminating exchange.

```
typedef struct _egress   /* Egress Service   */
{
    ElmtHdr eh;          /* element header   */
    TknStr  egress;      /* egress           */
} SiEgress;
```

Tokens

Token	ANSI 92	ANSI 95
egress	*	*

Event information

Associated variants: All except ANSI 88

Contains information sent in the backward direction to identify the type of event that caused a call progress message to be sent to the originating exchange.

```
typedef struct _evntInfo    /* Event Information                   */
{
    ElmtHdr eh;             /* element header                      */
    TknU8   evntInd;        /* event indicators                    */
    TknU8   evntPresResInd; /* event presentation restriction ind. */
} SiEvntInfo;
```

The fields in the SiEvntInfo structure are encoded as follows:

Field	Value
evntInd	Event indicators. Defined values: 0x01 = EV_ALERT Alerting 0x02 = EV_PROGRESS Progress 0x03 = EV_INBAND In-band information or an appropriate pattern is now available 0x04 = EV_FWDONBUSY Call forwarded on busy (national use) 0x05 = EV_FWDONNOREP Call forwarded on no reply (national use) 0x06 = EV_FWDUNCONDIT Call forwarded unconditional (national use) Additional values for ANSI 92: 0x08 = EV_NOTSUPPSERV Notification of supplementary services 0x06F= EV_SRVINFINC Service information included
evntPresResInd	Event presentation restriction indicator. Defined values: 0x00 = EVPR_NOIND No indication 0x01 = EVPR_PRESRES Presentation restricted

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 92	ANSI 95	BICC	ESTI V2	ESTI V3
evntInd	*	*	*	*	*
evntPresResInd	*	*	*	*	*

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White	ITU 97	Q.767
evntInd	*	*	*	*
evntPresResInd	*	*	*	*

Facility indicators

Associated variants: ANSI 88, BICC, ETSI V2, ETSI V3, ITU Blue, ITU White, ITU 97

Contains information sent in facility related messages in ITU-T networks.

```
typedef struct _facInd   /* Facility Indicators   */
{
    ElmtHdr eh;          /* element header        */
    TknU8   facInd;      /* facility indicator    */
} SiFacInd;
```

The facInd field is coded as follows:

0x02	FI_USR2USRSERV	User-to-user service.

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	BICC	ESTI V2	ETSI V3
facInd	*	*	*	*

Tokens for the ITU variants

Token	ITU Blue	ITU White	ITU 97
facInd	*	*	*

Facility information indicators

Associated variant: ANSI 88

Passes facility information in ANSI networks.

```
typedef struct _facInfInd       /* Facility Info Indicators       */
{
    ElmtHdr eh;                 /* element header                 */
    TknU8   calldPtyFreeInd;    /* called party free indicator    */
    TknU8   callgPtyAnsInd;     /* calling party answer ind.      */
    TknU8   facReqEnqInd;       /* facility request inquiry ind.  */
    TknU8   facReqActInd;       /* facility request active ind.   */
    TknU8   spare;              /* spare bits                     */
} SiFacInfInd;
```

The fields in the SiFacInfInd structure are encoded as follows:

Field	Value
	Called party free indicator. Defined values: 0x01 = CDPTY_BUSY Called party busy
callgPtyAnsInd	Called party answer indicator. Defined values: 0x00 = NOCGPTYANS No calling party answer 0x01 = CGPTYANS Calling party answer
	Facility request inquiry indicator. Defined values: 0x00 = NOENQUIRY No inquiry 0x01 = FACREQACTENQ Facility request active inquiry
facReqActInd	Facility request active indicator. Defined values: 0x00 = FACREQNOTACTIVE Facility request not active 0x01 = FACREQACTIVE Facility request active

Tokens

Token	ANSI 88
calldPtyFreeInd	*
callgPtyAnsInd	*
facReqEngInd	*
facReqActInd	*
spare	E-H

Forward call indicators

Associated variants: All

Contains information sent in an IAM message to notify the far exchange of the services required for a call.

```
typedef struct _fwdCallInd      /* Forward Call Indicators        */
{
    ElmtHdr eh;                 /* element header                 */
    TknU8   natIntCallInd;      /* National/Internat'l Call Ind.  */
    TknU8   end2EndMethInd;     /* end to end method indicator    */
    TknU8   intInd;             /* interworking indicator         */
    TknU8   segInd;             /* segmentation indicator         */
    TknU8   end2EndInfoInd;     /* end to end info indicator      */
    TknU8   isdnUsrPrtInd;      /* ISUP indicator                 */
    TknU8   isdnUsrPrtPrfInd;   /* ISUP preference ind.           */
    TknU8   isdnAccInd;         /* ISDN access indicator          */
    TknU8   sccpMethInd;        /* SCCP method indicator          */
    TknU8   spare;              /* spare bit                      */
    TknU8   natReserved;        /* reserved for national use      */
} SiFwdCallInd;
```

The fields in the SiFwdCallInd structure are encoded as follows:

Field	Value
natIntCallInd	National/international call indicator. Defined values: 0x00 = CALL_NAT Treat call as a national call 0x01 = CALL_INTERNAT Treat call as an international call
end2EndMethInd	End-to-end method indicator. Defined values for supported variants other than BICC: 0x00 = E2EMTH_NOMETH No end-to-end method available 0x01 = E2EMTH_PASSALNG Pass-along method available (national use) 0x02 = E2EMTH_SCCPMTH SCCP method available 0x03 = E2EMTH_BOTH Pass-along and SCCP methods available (national use) Defined values for BICC: 0x00 = E2EMTH_NOMETH No end-to-end method available ox01 = E2EMTH_RSVRD1 Reserved 0x02 = E2EMTH_RSVRD2 Reserved 0x03 = E2EMTH_RSVRD3 Reserved
intInd	Internetworking indicator. Defined values: 0x00 = INTIND_NOINTW No internetworking encountered. 0x01 = INTIND_INTW Internetworking encountered.
segInd	Segmentation indicator. Defined values: 0x00 = SEGIND_NOIND No indication 0x01 = SEGIND_INFO Additional information will be sent
end2EndInfoInd	End-to-end information indicator. Defined values for supported variants other than BICC: 0x00 = E2EINF_NOINFO No end-to-end information available 0x01 = E2EINF_INFO End-to-end information available Defined values for BICC: 0x00 = E2EINF_NOINFO No end-to-end information available 0x01 = E2EINF_RSVRD Reserved

Field	Value
isdnUsrPrtInd	ISUP/BICC indicator. Defined values for supported variants other than BICC: 0x00 = ISUP_NOTUSED ISDN user part not used all the way 0x01 = ISUP_USED ISDN user part used all the way 0x00 = BICC_NOTUSED BICC not used all the way 0x01 = BICC_USED BICC used all the way
isdnUsrPrtPrfInd	ISUP/BICC preference indicator. Defined values: 0x00 = PREF_PREFAW Preferred all the way 0x01 = PREF_NOTREQAW Not required all the way 0x02 = PREF_REQAW Required all the way
isdnAccInd	ISDN access indicator. Defined values: 0x00 = ISDNACC_NONISDN Originating access non-ISDN 0x01 = ISDNACC_ISDN Originating access ISDN
sccpMethInd	SCCP method indicator. Defined values for supported variants other than BICC: 0x00 = SCCPMTH_NOIND No indication 0x01 = SCCPMTH_CONLESS Connectionless model available (national use) 0x02 = SCCPMTH_CONORNTD Connection oriented method available 0x03 = SCCPMTH_BOTH Connectionless and connection oriented methods available (national use) Defined values for BICC: 0x00 = SCCPMTH_NOIND No indication 0x01 = SCCPMTH_RSVRD1 Reserved 0x02 = SCCPMTH_RSVRD2 Reserved 0x03 = SCCPMTH_RSVRD3 Reserved

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88		ANSI 95	BICC	ETSI V2	ETSI V3
natIntCallInd	*	*	*	*	*	*
end2EndMethInd	*	*	*	*	*	*
intInd	*	*	*	*	*	*
segInd		*	*			
end2EndInfoInd	*			*	*	*
isdnUsrPrtInd	*	*	*	*	*	*
isdnUsrPrtPrfInd	*	*	*	*	*	*
isdnAccInd	*	*	*	*	*	*
sccpMethInd		*	*	*	*	*
spare	J-L	L	L	L	L	L
natReserved	M-P	M-P	M-P	M-P	M-P	M-P

Tokens for the ITU and Q.767 variants

Token	ITU Blue		ITU 97	Q.767
natIntCallInd	*	*	*	*
end2EndMethInd	*	*	*	*
intInd	*	*	*	*
segInd				
end2EndInfoInd	*	*	*	*
isdnUsrPrtInd	*	*	*	*
isdnUsrPrtPrfInd	*	*	*	*
isdnAccInd	*	*	*	*
sccpMethInd	*	*	*	*
spare	L	L	L	L
natReserved	M-P	M-P	M-P	M-P

Forward GVNS

Associated variants: BICC, ITU White

Conveys global virtual network service-related information in the forward direction. Forward GVNS is an extended element. Refer to *Extended element* on page 185 for more information.

Free phone indicators

Associated variant: ETSI V2

Provides the free phone indicators format.

```
typedef struct _freePhnInd     /* Free Phone Indicators  */
{
    ElmtHdr eh;                /* element header         */
    TknU8   freeInd;           /* Free phone indicator   */
    TknU8   spare1;            /* bits B-H               */
}SiFreePhnInd;
```

The freeInd field is encoded to one of the following values:

0	FREE_PHN_NO_INDICATION
1	FREE_PHN_CALL

Tokens

Token	ETSI V2
freeInd	*
spare1	B-H

Generic address

Associated variants: ANSI 92, ANSI 95

Identifies the type of address, numbering plan, and actual address presented in a call setup.

```
typedef struct _genAddr   /* Generic Address              */
{
    ElmtHdr eh;           /* element header               */
    TknU8   typeOfAddr;   /* type of address              */
    TknU8   natAddr;      /* nature of address indicator  */
    TknU8   oddEven;      /* odd or even address signal   */
    TknU8   reserved;     /* reserved for national use    */
    TknU8   presRest;     /* presentation restriction     */
    TknU8   numPlan1;     /* numbering plan               */
    TknU8   spare;        /* spare bits                   */
    TknStr  addrSig;      /* addressing signal            */
} SiGenAddr;
```

The fields in the SiGenAddr structure are encoded as follows:

Field	Value
typeOfAddr	Type of address. Defined values: 0x00 = DIALNUM 0x01 = DESTNUM 0x02 = SUPADDR_FAIL 0x03 = SUPADDR_NOTSCREEN 0x04 = COMPLNUM
natAddr	Nature of address indicator. Defined values for all variants: 0x03 = NATNUM Nationally significant number 0x04 = INTNATNUM International number Additional values for ANSI: 0x71 = SUBSNUMOPREQ Subscriber number operator requested 0x72 = NATNUMOPREQ National number operator requested 0x73 = INTNATNUMOPREQ International number operator requested 0x74 = NONUMPRESOPREQ No number present operator requested 0x75 = NONUMPRESCUTTHRU No number present cut-through call to carrier 0x76 = NINEFIVEOH 950+ service 0x77 = TSTLINETSTCODE Test line test code
oddEven	Specifies whether the number of digits in the addrSig field is even or odd. If even, the last octet contains two digits. If odd, the last octet contains only one digit and the most significant four bits not used. Defined values: 0 = NMB_EVEN 1 = NMB_ODD
Reserved	Reserved bits.
presRest	Presentation restriction. Defined values: 0x00 = PRESALLOW Presentation allowed 0x01 = PRESREST Presentation restricted 0x02 = ADDRNOAVAIL Address not available
numPlan1	Numbering plan. Defined values for all supported variants except BICC: 0x00 = NP_UNK Unknown 0x01 = NP_ISDN ISDN/telephony according to E.164/E.163 0x02 = NP_TEL Telephony numbering according to E.163 0x03 = NP_DATA Data numbering according to X.121 0x04 = NP_TELEX Telex number according to F.69 0x08 = NP_NATIONAL National standard numbering 0x09 = NP_PRIVATE Private numbering plan 0x0f = NP_EXT Reserved for extension
Spare	Spare bits.
addrSig	Actual address digits, encoded as shown in the following tables.

For the addrSig field, the actual address digits are encoded as follows:

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

where each digit is encoded with the following bit pattern:

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

Tokens

Token	ANSI 92	ANSI 95
typeOfAddr	*	*
natAddr	*	*
oddEven	*	*
reserved	3(1-2)	3(1-2)
presRest	*	*
numPlan1	*	*
spare	3(8)	3(8)
addrSig	*	*

Generic digits

Associated variants: ANSI 92, ANSI 95, BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Provides additional numeric data associated with supplemental services such as authorization code, PIN number, or account code.

```
typedef struct _genDigits    /* Generic Digits   */
{
    ElmtHdr eh;              /* element header   */
    TknU8   typeOfDigits;    /* type of digits   */
    TknU8   encodeScheme;    /* encoding scheme  */
    TknStr  digits;          /* digits           */
} SiGenDigits;
```

The fields in the SiGenDigits structure are encoded as follows:

Field	Value
typeOfDigits	Type of digits. Defined values: 0x00 = ACCTCODE Account code 0x00 = AUTHCODE Authorization code 0x02 = PRIVNETMARK Private networking traveling class mark 0x03 = BUSCOMMGRID Business communication group identity
encodeScheme	Encoding scheme. Defined values: 0x00 = ENC_BCD_EVEN Even number of digits 0x01 = ENC_BCD_ODD Odd number of digits 0x02 = ENC_IA5 IA5 character 0x03 = ENC_BIN Binary coded
digits	Digits are encoded as shown in the following tables.

For the addrSig field, the actual address digits are encoded as follows:

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

where each digit is encoded with the following bit pattern:

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

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 92	ANSI 95	BICC	ETSI V2	ETSI V3
typeOfDigits	*	*	*	*	*
encodeScheme	*	*	*	*	*
digits	*	*	*	*	*

Tokens for the ITU variants

Token	ITU White	ITU 97
typeOfDigits	*	*
encodeScheme	*	*
digits	*	*

Generic name

Associated variant: ANSI 95

Provides name data associated with supplemental services.

```
typedef struct _genName       /* Generic Name              */
{
    ElmtHdr eh;               /* element header            */
    TknU8   presRest;         /* presentation restriction */
    TknU8   spare;            /* spare bits                */
    TknU8   availability;     /* name availability         */
    TknU8   type;             /* type of name              */
    TknU8   name;             /* name                      */
} SiGenName;
```

The fields in the SiGenName structure are encoded as follows:

Field	Value
presRest	Presentation restriction. Defined values: 0x00 = PRESALLOW Presentation allowed 0x01 = PRESREST Presentation restricted 0x02 = PRESBLKTGL Blocking toggle 0x03 = PRESNOIND No indication
spare	Spare bits.
availability	Name availability. Defined values: 0x00 = GNA_AVAIL Name available/unknown 0x01 = GNA_NOTAVAIL Name not available
type	Name type. Defined values: 0x01 = GNT_CALLING Calling name 0x02 = GNT_ORIGCALLED Original called name 0x03 = GNT_REDIRECTING Redirecting name
name	Encoded as a 1 to 15 character ASCII string.

Tokens

Token	ANSI 95
presRest	*
spare	1(3-4)
availability	*
type	*
name	*

Generic number

Associated variants: BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Represents a number passed in either direction for enhanced network operation or supplementary services.

```
typedef struct _genNum    /* Generic Number                         */
{
    ElmtHdr eh;           /* element header                         */
    TknU8   nmbQual;      /* number qualifier                       */
    TknU8   natAddrInd;   /* nature of address indicator            */
    TknU8   oddEven;      /* odd or even                            */
    TknU8   scrnInd;      /* screen indicator                       */
    TknU8   presRest;     /* Addr presentation restricted indicator */
    TknU8   numPlan;      /* numbering plan                         */
    TknU8   niInd;        /* number incomplete indicator            */
    TknStr addrSig;       /* Address Signal                         */
} SiGenNum;
```

The fields in the SiGenNum structure are encoded as follows:

Field	Value
nmbQual	Number qualifier. Defined values: 0x05 = NQ_ADDCONMNB Additional connected number 0x06 = NQ_ADDCGNMB Additional calling party number 0x07 = NQ_ORIGCDNMB Additional original called party number 0x08 = NQ_ORIGRGDNMB Additional redirecting number 0x09 = NQ_ORIGRDNMB Additional redirection number
natAddrInd	Nature of address indicator. Defined values for all supported variants except ANSI: 0x01 = SUBSNUM Subscriber number 0x04 = INTNATNUM International number Additional values for ANSI: 0x71 = SUBSNUMOPREQ Subscriber number, operator requested 0x72 = NATNUMOPREQ National number, operator requested 0x74 = NONUMPRESOPREQ No number present, operator requested 0x75 = NONUMPRESCUTTHRU No number present, cut-through call to carrier 0x77 = TSTLINETSTCODE Test line test code 0x76 = NINEFIVEOH 950+ service
	Specifies whether the number of digits in the addrSig field is even or odd. If even, the last octet contains two digits. If odd, the last octet contains only one digit and the most significant four bits not used. Defined values: 0 = NMB_EVEN 1 = NMB_ODD
scrnInd	Screen indicator. Defined values: 0x00 = USRPROVNOTVER User provided, not verified 0x01 = USRPROV User provided, verified passed 0x02 = USRPROVVERFAIL User provided, verified failed 0x03 = NETPROV Network provided
presRest	Address presentation restricted indicator. Defined values: 0x00 = PRESALLOW Presentation allowed 0x01 = PRESREST Presentation restricted 0x02 = ADDRNOAVAIL Address not available

Field	Value
numPlan	Numbering plan. Defined values for all supported variants except BICC: 0x00 = NP_UNK Unknown 0x01 = NP_ISDN ISDN/telephony according to E.164/E.163 0x02 = NP_TEL Telephony numbering according to E.163; spare in ITU White and ITU 97 0x03 = NP_DATA Data numbering according to X.121 0x04 = NP_TELEX Telex number according to F.69 0x08 = NP_NATIONAL National standard numbering 0x09 = NP_PRIVATE Private numbering plan 0x0f = NP_EXT Reserved for extension Defined values for BICC: 0x00 = NP_UNK Unknown 0x01 = NP_ISDN ISDN/telephony according to ITU-T E.164/E.163 0x02 = NP_TEL Spare 0x03 = NP_DATA Data numbering according to ITU-T X.121 0x04 = NP_TELEX Telex number according to ITU_T F.69 0x05 = NP_PRIVATE Private numbering plan. 0x06 = NP_NATIONAL Reserved for national use
niInd	Number incomplete indicator. Defined values: 0x00 = NBMCMLTE Number complete 0x01 = NBMINCMLTE Number incomplete
addrSig	Actual address digits, encoded as shown in the following tables.

For the addrSig field, the actual address digits are encoded as follows:

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

where each digit is encoded with the following bit pattern:

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

Tokens

Token	BICC	ETSI V2	ETSI V3	ITU White	ITU 97
nmbQua1	*	*	*	*	*
natAddrInd	*	*	*	*	*
oddEven	*	*	*	*	*
scrnInd	*	*	*	*	*
presRest	*	*	*	*	*
numPlan	*	*	*	*	*
niInd	*	*	*	*	*
addrSig	*	*	*	*	*

Hop counter

Associated variants: ANSI 95, BICC

Provides the hop counter format.

```
typedef struct _hopcount  /* Hop Counter      */
{
    ElmtHdr eh;           /* element header   */
    TknU8   hopCount;     /* hop count        */
    TknU8   spare;        /* spare bits       */
} SiHopCount;
#endif
```

Tokens

Token	ANSI 95	BICC
hopCount	*	*
spare	1(6-8)	F-H

Index

Associated variant: ANSI 88

Provides the index format.

```
typedef struct _index   /* Index           */
{
    ElmtHdr eh;         /* element header  */
    TknU32  index;      /* index           */
} SiIndex;
#endif
```

Tokens

Token	ANSI 88
index	*

Information indicators

Associated variants: All

Provides the far exchange with additional information about a call in progress.

```
typedef struct _infoInd      /* Information Indicators                    */
{
    ElmtHdr eh;                /* element header                            */
    TknU8   cgPtyAddrRespInd;  /* calling party address response indicator */
    TknU8   holdProvInd;       /* hold provided indicator                   */
    TknU8   spare1;            /* spare bits                                */
    TknU8   cgPtyCatRespInd;   /* calling party category response indicator */
    TknU8   chrgInfoRespInd;   /* charge information response ind.          */
    TknU8   solInfoInd;        /* solicitation information ind.             */
    TknU8   connAddrRspInd;    /* connected addr response ind.              */
    TknU8   redirAddrRspInd;   /* redirection address response indicator    */
    TknU8   indexRspInd;       /* index response indicator                  */
    TknU8   spare2;            /* spare bits                                */
    TknU8   mlbgInfoInd;       /* multi location business group information */
                               /* indicator                                 */
    TknU8   reserved;          /* reserved                                  */
} SiInfoInd;
```

The fields in the SiInfoInd structure are encoded as follows:

Field	Value
	Calling party address response indicator. Defined values for all supported variants except ANSI: 0x00 = CGPRTYADDRESP_NOTINCL Calling party address not included 0x01 = CGPRTYADDRESP_NOTAVAIL Calling party address not available 0x03 = CGPRTYADDRESP_INCL Calling party address included Defined values for ANSI: 0x03 = CGPTYADDRSPINCLNOHOLD Calling party address included, hold not provided 0x04 = CGPTYADDRSPINCLHOLD Calling party address included, hold provided
holdProvInd	Hold provided indicator. Defined values: 0x00 = HOLD_NOTPROV Hold not provided 0x01 = HOLD_PROV Hold provided
cgPtyCatRespInd	Calling party category response indicator. Defined values: 0x00 = CGPRTYCATRESP_NOTINCL Calling party's response not included 0x01 = CGPRTYCATRESP_INCL Calling party's response included
chrgInfoRespInd	Charge information response indicator. Defined values 0x00 = CHRGINFO_NOTINCL Charge information not included 0x01 = CHRGINFO_INCL Charge information included
solInfoInd	Solicitation information indicator. Defined values: 0x00 = SOLINFO_SOLICIT Solicited 0x01 = SOLINFO_UNSOLICIT Unsolicited
connAddrRspInd	Solicitation information indicator. Defined values: 0x00 = CONNADDRNOTINCL Connected address not included 0x01 = CONNADDRNOTAVAIL Connected address not available 0x03 = CONNADDRINCL Connected address included
redirAddrRspInd[a|b]	Redirection address response indicator. Defined values: 0x00 = REDIRGADDRNOTINCL Redirecting address not included 0x01 = REDIRGADDRNOTAVAIL Redirecting address not available 0x03 = REDIRGADDRINCL Redirecting address included
indexRspInd	Index response indicator. Defined values: 0x00 = INDEXNOTINCL Index not included 0x01 = INDEXINCL Index included
mlbgInfoInd	Multi location business group information. Defined values: 0x00 = MLBGINFOTINCL Multi-location business group information not included 0x01 = MLBGINFOINCL Multi-location business group information included

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	ANSI 92		BICC	ETSI V2	ETSI V3
cgPtyAddrRespInd	*	*	*	*	*	*
holdProvInd		*	*	*	*	*
spare1	K-P	D-E	D-E	D-E	D-E	D-E
cgPtyCatRespInd	*	*	*	*	*	*
chrgInfoRespInd	*	*	*	*	*	*
solInfoInd		*	*	*	*	*
connAddrRspInd	*	*	*			
redirAddrRspInd	*					
indexRspInd	*					
spare2		I-O	I-O	I-L	I-L	I-L
mlbgInfoInd		*	*			
reserved				M-P	M-P	M-P

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White	ITU 97	Q.767
cgPtyAddrRespInd	*	*	*	
holdProvInd	*	*	*	
spare1	D-E	D-E	D-E	
cgPtyCatRespInd	*	*	*	
chrgInfoRespInd	*	*	*	
solInfoInd	*	*	*	
connAddrRspInd				
redirAddrRspInd				
indexRspInd				
spare2	I-P	I-L	I-L	
mlbgInfoInd				
reserved		M-P	M-P	

Information request indicators

Associated variants: All except Q.767

```
typedef struct _infoReqInd   /* Information Request Indicators */
{
    ElmtHdr eh;               /* element header */
    TknU8   cgPtyAdReqInd;    /* calling party address request ind. */
    TknU8   holdingInd;       /* holding indicator */
    TknU8   spare1;           /* spare */
    TknU8   cgPtyCatReqInd;   /* calling party category request ind. */
    TknU8   chrgInfoReqInd;   /* charge information request indicator */
    TknU8   malCaIdReqInd;    /* malicious call id request indicator */
    TknU8   mlbgInfoInd;      /* multi location business group information indicator */
    TknU8   connAddrReqInd;   /* connected address request indicator */
    TknU8   redirAddrReqInd;  /* redirection address request indicator */
    TknU8   indexReqInd;      /* index request indicator */
    TknU8   spare2;           /* spare */
    TknU8   spare3;           /* spare */
    TknU8   reserved;
} SiInfoReqInd;
```

The fields in the SiInfoReqInd structure are encoded as follows:

Field	Value
cgPtyAdReqInd	Calling party address request indicator. Defined values: 0x00 = CGPRTYADDREQ_NOTREQ Calling party address not requested 0x01 = CGPRTYADDREQ_REQ Calling party address requested
holdingInd	Holding indicator. Defined values: 0x00 = HOLD_NOTREQD Holding not requested 0x01 = HOLD_REQD Holding requested
cgPtyCatReqInd	Calling party category request indicator. Defined values for supported variants other than ANSI: 0x00 = CGPRTYCATREQ_NOTREQ Calling party's category not requested 0x01 = CGPRTYCATREQ_REQ Calling party's category requested Defined values for ANSI: 0x03 = CGPTYADDRSPINCLNOHOLD Calling party address included, hold not provided 0x04 = CGPTYADDRSPINCLHOLD Calling party address included, hold provided
chrgInfoReqInd	Charge information request indicator. Defined values: 0x00 = CHRGINFO_NOTREQ Charge information not requested 0x01 = CHRGINFO_REQ Charge information requested
malCaIdReqInd	Malicious call ID request indicator. Defined values: 0x00 = MALCAID_NOTREQ Malicious call identification not requested 0x01 = MALCAID_REQ Malicious call identification requested

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	ANSI 92	ANSI 95	BICC	ETSI V2	ETSI V3
cgPtyAdReqInd	*	*	*	*	*	*
holdingInd		*	*	*	*	*
spare1	K-P	D-E	D-E	F-G	D-E	D-E
cgPtyCatReqInd	*	*	*	*	*	*
chrgInfoReqInd	*	*	*	*	*	*
malCaIdReqInd		*	*	*	*	*
mlgbInfoInd		*	*			
spare2		I-O	I-O	I-L		
spare3				I-L	I-L	I-L

Tokens for the ITU variants

Token	ITU Blue	ITU White	ITU 97
cgPtyAdReqInd	*	*	*
holdingInd	*	*	*
spare1	D-E	D-E	D-E
cgPtyCatReqInd	*	*	*
chrgInfoReqInd	*	*	*
malCaIdReqInd	*	*	*
mlgbInfoInd			
spare2	I-P	I-L	I-L
spare3	I-P		

Jurisdiction information

Associated variants: ANSI 92, ANSI 95

Provides numeric data indicating the geographic origination of a call.

```
typedef struct _jurisInf  /* Jurisdiction Info */
{
    ElmtHdr eh;           /* element header    */
    TknU8   addrSig1;     /* address signal 1  */
    TknU8   addrSig2;     /* address signal 2  */
    TknU8   addrSig3;     /* address signal 3  */
    TknU8   addrSig4;     /* address signal 4  */
    TknU8   addrSig5;     /* address signal 5  */
    TknU8   addrSig6;     /* address signal 6  */
} SiJurisInf;
```

Refer to *Called party number* on page 201 for information on encoding address signal digits.

Tokens

Token	ANSI 92	ANSI 95
addrSig1	*	*
addrSig2	*	*
addrSig3	*	*
addrSig4	*	*
addrSig5	*	
addrSig6	*	*

Location number

Associated variants: BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Provides the location number format.

```
typedef struct _locNum    /* Location Number                */
{
    ElmtHdr eh;           /* element header                 */
    TknU8   natAddrInd;   /* nature of address indicator    */
    TknU8   oddEven;      /* odd or even                    */
    TknU8   scrnInd;      /* screen indicator               */
    TknU8   presRest;     /* Addr presentation restricted   */
    TknU8   numPlan;      /* numbering plan                 */
    TknU8   niInd;        /* number incomplete indicator    */
    TknU8   addrSig;      /* Address Signal                 */
```

The fields in the SiLocNum structure are encoded as follows:

Field	Value
natAddrInd	Nature of address indicator. Defined values for all supported variants: 0x01 = SUBSNUM Subscriber number 0x03 = NATNUM Nationally significant number 0x04 = INTNATNUM International number Additional values for ANSI: 0x71 = SUBSNUMOPREQ Subscriber number operator requested 0x72 = NATNUMOPREQ National number operator requested 0x73 = INTNATNUMOPREQ International number operator requested 0x74 = NONUMPRESOPREQ No number present operator requested 0x75 = NONUMPRESCUTTHRU No number present cut-through call to carrier 0x77 = TSTLINETSTCODE Test line test code 0x76 = NINEFIVEOH 950+ service
oddEven	Specifies whether the number of digits in the addrSig field is even or odd. If even, the last octet contains two digits. If odd, the last octet contains only one digit and the most significant four bits not used. Defined values: 0 = NMB_EVEN 1 = NMB_ODD
scrnInd	Screen indicator. Defined values: 0x00 = USRPROVNOTVER User provided, not verified 0x01 = USRPROV User provided, verified passed 0x02 = USRPROVVERFAIL User provided, verified failed 0x03 = NETPROV Network provided
presRest	Address presentation restricted. Defined values: 0x00 = PRESALLOW Presentation allowed 0x01 = PRESREST Presentation restricted 0x02 = ADDRNOAVAIL Address not available
numPlan	Numbering plan. Defined values for all supported variants except BICC: 0x00 = NP_UNK Unknown 0x01 = NP_ISDN ISDN/telephony according to E.164/E.163 0x02 = NP_TEL Telephony numbering according to E.163; spare in ITU White and ITU 97 0x03 = NP_DATA Data numbering according to X.121 0x04 = NP_TELEX Telex number according to F.69 0x08 = NP_NATIONAL National standard numbering 0x09 = NP_PRIVATE Private numbering plan 0x0f = NP_EXT Reserved for extension Defined values for BICC: 0x00 = NP_UNK Unknown 0x01 = NP_ISDN ISDN/telephony according to ITU-T E.164/E.163 0x02 = NP_TEL Spare 0x03 = NP_DATA Data numbering according to ITU-T X.121 0x04 = NP_TELEX Telex number according to ITU_T F.69 0x05 = NP_PRIVATE Private numbering plan. 0x06 = NP_NATIONAL Reserved for national use
niInd	Number complete indicator. Defined values: 0x00 = NBMCMLTE Number complete 0x01 = NBMINCMLTE Number incomplete
addrSig	Actual address digits, encoded as shown in the following tables.

For the addrSig field, the actual address digits are encoded as follows:

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

where each digit is encoded with the following bit pattern:

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

Tokens

Token	BICC	ETSI V2	ETSI V3	ITU White	ITU 97
natAddrInd	*	*	*	*	*
oddEven	*	*	*	*	*
scrnInd	*	*	*	*	*
presRest	*	*	*	*	*
numPlan	*	*	*	*	*
niInd	*	*	*	*	*
addrSig	*	*	*	*	*

Loop prevention indicator

Associated variants: BICC, ETSI V2, ITU White, ITU 97

Provides the loop prevention indicator format.

```
typedef struct _loopPrevInd /* Loop Prevention Indicator                */
{
    ElmtHdr eh;           /* element header                             */
    TknU8   loopTypeInd;  /* Type indicator                             */
    TknU8   responseInd;  /* always 7 bits, B-H: all spare if loopType=0 */
}SiLoopPrevInd;
```

The fields in the SiLoopPrevInd structure are encoded as follows:

Field	Value
loopTypeInd	Type indicator. Defined values: 0 = LOOP_TYPE_REQUEST Request 1 = LOOP_TYPE_RESPONSE Response
responseInd	Response indicator. Defined values if loopTypeInd is 1: 0 = LOOP_RESP_INSUFF Insufficient information 1 = LOOP_RESP_NO_LOOP No loop exists 2 = LOOP_RESP_SIMULTANEOUS Simultaneous transfer 3 = LOOP_RESP_SPARE Spare

Tokens

Token	BICC	ETSI V2	ITU White	
loopTypeInd	*	*	*	*
responseInd	B-H	B-H	B-H	B-H

MCID request

Associated variants: BICC, ITU White

Contains information sent in the backward direction to request identification of the calling party for the purpose of malicious call identification.

```
typedef struct _mcidReq   /* MCID request indicators      */
{
    ElmtHdr eh;           /* element header               */
    TknU8   reqInd;       /* mcid request indicators      */
    TknU8   hldInd;       /* hold indicators              */
    TknU8   spare;        /* spare bits                   */
} SiMcidReqInd;
```

The fields in the SiMcidReqInd structure are encoded as follows:

Field	Value
reqInd	MCID request indicator. Defined values: 0x00 = MALCAID_NOTREQ MCID not requested 0x01 = MALCAID_REQ MCID requested
hldInd	Hold indicator. Defined values: 0x00 = HOLD_NOTREQ Holding not requested 0x01 = HOLD_REQ Holding requested

Tokens

Token	BICC	ITU White
reqInd	*	*
hldInd	*	*
spare	C-H	C-H

MCID response

Associated variants: BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Contains information sent as a response to a malicious call identification request.

```
typedef struct _mcidRsp  /* MCID response indicators */
{
    ElmtHdr eh;          /* element header           */
    TknU8   rspInd;      /* mcid response indicators */
    TknU8   hldInd;      /* hold indicators          */
    TknU8   spare;       /* spare bits               */
} SiMcidRspInd;
```

The fields in the SiMcidRspInd structure are encoded as follows:

Field	Value
rspInd	MCID response indicators. Defined values: 0x00 = MCID_NOTINCLDD: MCID not included 0x01 = MCID_INCLDD MCID included
hldInd	Hold indicators. Defined values: 0x00 = HOLD_NOTPROV Holding not provided 0x01 = HOLD_PROV Holding provided

Tokens

Token	BICC	ETSI V2	ETSI V3	ITU White	ITU 97
rspInd	*	*	*	*	*
hldInd	*	*	*	*	*
spare	C-H	C-H	C-H	C-H	C-H

Message compatibility

Associated variants: BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Contains information sent in either direction to instruct the far exchange on what to do if this message is unrecognized. NaturalAccess™ ISUP supports up to two instruction indicators.

```
typedef struct _msgCom    /* message compatibility information */
{
    ElmtHdr eh;           /* element header                    */
    TknU8   tranXInd;     /* transit exchange ind.             */
    TknU8   relCllInd;    /* release call indicator            */
    TknU8   sndNotInd;    /* send notification indicator       */
    TknU8   spare;        /* spare bits                        */
    TknU8   dcrdMsgInd;   /* discard message indicator         */
    TknU8   passNotPoss;  /* pass on not possible ind.         */
    TknU8   tranXInd1;    /* transit exchange indicator        */
    TknU8   relCllInd1;   /* release call indicator            */
    TknU8   sndNotInd1;   /* send notification indicator       */
    TknU8   spare1;       /* spare bits                        */
    TknU8   dcrdMsgInd1;  /* discard message indicator         */
    TknU8   passNotPoss1; /* pass on not possible ind.         */
} SiMsgCompInfo;
```

For the BICC variant, the following structure is used for sending information regarding unrecognized messages. The application should not use both structures at the same time.

```
typedef struct _biccMsgCom           /* message compatibility information for ANSI and
ITU BICC */
{
   ElmtHdr eh;                     /* element header */
   TknU8   tranXInd;               /* transit exchange indicator */
   TknU8   relCllInd;              /* release call indicator */
   TknU8   sndNotInd;              /* send notification indicator */
   TknU8   dcrdMsgInd;             /* discard message indicator */
   TknU8   passNotPoss;            /* pass on not possible indicator */
   TknU8   bnIntwInd;              /* broadband/narrowband interworking indicator */
   TknU8   tranXInd1;              /* transit exchange indicator */
   TknU8   relCllInd1;             /* release call indicator */
   TknU8   sndNotInd1;             /* send notification indicator */
   TknU8   dcrdMsgInd1;            /* discard message indicator */
   TknU8   passNotPoss1;           /* pass on not possible indicator */
   TknU8   bnIntwInd1;             /* broadband/narrowband interworking indicator */
} SiBiccMsgCompInfo;
```

The fields in the SiMsgCompInfo structure are encoded as follows:

Field	Value
tranXInd[1]	Transit exchange indicator. Defined values: 0x00 = Transit exchange interpretation 0x01 = End node interpretation
relClIInd[1]	Release call indicator. Defined values: 0x00 = Do not release call 0x01 = Release call
sndNotInd[1]	Send notification indicator. Defined values: 0x00 = Do not send notification 0x01 = Send notification (confusion message)
dcrdMsgInd[1]	Discard message indicator. Defined values: 0x00 = Do not discard (pass on) message 0x01 = Discard message
passNotPoss[1]	Pass-on not possible indicator. Defined values: 0x00 = Release call if pass-on not possible 0x01 = Discard information if pass-on not possible
bnIntwInd[1]	Broadband/narrowband interworking indicator. Defined values: 0x00 = Pass on 0x01 = Discard message 0x02 = Release call 0x03 = reserved

Tokens

Token	BICC	ETSI V2	ETSI V3	ITU White	ITU 97
tranXInd	*	*	*	*	*
relClIInd	*	*	*	*	*
sndNotInd	*	*	*	*	*
spare		F-G	F-G	F-G	F-G
dcrdMsgInd	*	*	*	*	*
passNotPoss	*	*	*	*	*
bnIntwInd	*				
tranXInd1	*	*	*	*	*
relClIInd1	*	*	*	*	*
sndNotInd1	*	*	*	*	*
spare1		F-G	F-G	F-G	F-G
dcrdMsgInd1	*	*	*	*	*
passNotPoss1	*	*	*	*	*
bnIntwInd1	*				

MLPP precedence

Associated variants: ANSI 95, BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Specifies the caller's MLPP precedence level and service domain.

```
typedef struct _mlppPrec   /* MLPP precedence   */
{
    ElmtHdr eh;            /* element header    */
    TknU8   precdLvl;      /* precedence level  */
    TknU8   spare1;        /* spare bits        */
    TknU8   lfb;           /* LFB               */
    TknU8   spare2;        /* spare bits        */
    TknU8   frstDig;       /* first digit       */
    TknU8   scndDig;       /* second digit      */
    TknU8   thrdDig;       /* third digit       */
    TknU8   frthDig;       /* fourth digit      */
    TknU32  servDomain;    /* service domain    */
} SiMlppPrec;
```

The fields in the SiMlppPrec structure are encoded as follows:

Field	Value
precdLvl	Precedence level. Defined values: 0x00 = PL_FLASHORD Flash override 0x01 = PL_FLASH Flash 0x02 = PL_IMMDT Immediate 0x03 = PL_PRIOR Priority 0x04 = PL_ROUTINE Routine
spare1	Spare bits.
lfb	Look ahead for busy (LFB). Defined values: 0x00 = LFB_ALLWD: Look ahead for busy (LFB) allowed 0x01 = LFB_PTHRSRVD Path reserved (national use) 0x02 = LFB_NOTALLWD LFB not allowed
spare2	Spare bits.
frstDig scndDig thrdDig frthDig	Four digits (binary representation) of the network identity code (0 + telephone country code). Refer to ITU-T Recommendation Q.763 for more information.
servDomain	Network-specific service domain. Only low order 24 bits are used.

Tokens

Token	ANSI 95	BICC	ETSI V2	ETSI V3	ITU White	ITU 97
precdLvl	*	*	*	*	*	*
spare1		1(5)	1(5)	1(5)	1(5)	1(5)
lfb	*	*	*	*	*	*
spare2		1(8)	1(8)	1(8)	1(8)	1(8)
frstDig	*	*	*	*	*	*
scndDig	*	*	*	*	*	*
thrdDig	*	*	*	*	*	*
frthDig	*	*	*	*	*	*
servDomain	*	*	*	*	*	*

Nature of connection indicators

Associated variants: All

Contains information sent in the forward direction with information regarding the circuit connection desired to enable intermediate exchanges to determine how to process a call.

```
typedef struct _natConInd      /* Nature of Connection Ind.     */
{
    ElmtHdr eh;                /* element header                */
    TknU8   satInd;            /* Satellite Indicator           */
    TknU8   contChkInd;        /* continuity check indicator    */
    TknU8   echoCntrlDevInd;   /* echo control device indicator */
    TknU8 spare;
 SiNatConInd;
```

The fields in the SiNatConInd structure are encoded as follows:

Field	Value
satInd	Satellite indicator. Defined values: 0x00 = SAT_NONE: No satellite circuit in the connection 0x01 = SAT_ONE One satellite circuit in the connection 0x02 = SAT_TWO Two satellite circuits in the connection
contChkInd	Continuity check indicator. Defined values for all supported variants except BICC: 0x00 = CONTCHK_NOTREQ Continuity check not required 0x01 = CONTCHK_REQ Continuity check not required on this circuit 0x02 = CONTCHK_PREV Continuity check performed on a previous circuit 0x03 = CONTCHK_SPARE Spare Defined values for BICC: 0x00 = CONT_NOTEXP No COT expected 0x01 = CONT_RSRVD Reserved 0x02 = CONT_EXP COT expected
echoCntrlDevInd	Echo control device indicator. Defined values: 0x00 = ECHOCDEV_NOTINCL Outgoing echo control device not included. 0x01 = ECHOCDEV_INCL Outgoing echo control device included

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	ANSI 92	ANSI 95	BICC	ETSI V2	ETSI V3
satInd	*	*	*	*	*	*
contChkInd	*	*	*	*	*	*
echoCntrlDevInd	*	*	*	*	*	*
spare	F-H	F-H	F-H	F-H	F-H	F-H

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White	ITU 97	Q.767
satInd	*	*	*	*
contChkInd	*	*	*	*
echoCntrlDevInd	*	*	*	*
spare	F-H	F-H	F-H	F-H

Network management controls

Associated variants: BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Provides the network management controls format.

```
typedef struct _netMngmtCtrls  /* Network Management Controls           */
{
    ElmtHdr eh;                /* element header                        */
    TknU8   tarInd;            /* Temporary Alternate Routing indicator */
    TknU8   spare1;            /* bits B-G                              */
}SiNetMngmtCtrls;
```

The tarInd field is encoded to one of the following values:

0	TAR_IND_NO_INDICATION	No indication.
1	TAR_IND_CONTROLLED_CALL	Tar controlled call.

Tokens

Token	BICC	ETSI V2	ETSI V3	ITU White	ITU 97
tarInd	*	*	*	*	*
spare1	B-G	B-G	B-G	B-G	B-G

Network specific facility

Associated variants: BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Transparently transfers service-related information between a local exchange and the identified network.

```
typedef struct _netFac    /* network specific facility   */
{
    ElmtHdr eh;           /* element header              */
    TknStr  netFac;       /* network facility1           */
} SiNetSpecFacil;
```

The netFac field is explained in the ITU-T Recommendation Q.763 (1993).

Tokens

Token	BICC	ETSI V2		ITU White	ITU 97
netFac	*	*	*	*	*

Network transport

Associated variants: ANSI 92, ANSI 95

Transports non-standard ISUP information elements.

```
typedef struct _netTransport   /* Network Transport       */
{
    ElmtHdr eh;                /* element header          */
    TknStr  netTransport;      /* network transport       */
} SiNetTransport;
```

The netTransport field is encoded as IE name and IE length, followed by IE contents.

Tokens

Token	ANSI 92	
netTransport	*	*

Notification indicator

Associated variants: All except ANSI 88 and ITU Blue

Provides information regarding supplementary services such as Centrex services.

Note: This information element is called Generic notification indicator in the ITU and BICC variants.

```
Typedef struct _notifInd  /* Notification Indicator */
{
    ElmHdr eh             /* element header         */
    TknU8  notifInd       /* Notification Indicator */
} SiNotifInd;
```

The notifInd field is encoded as follows:

0x04	NI_CALLDELAY	Call completion delay.
0x60	NI_CALLWAIT	
0x79	NI_REMHLD	Remote hold.
0x7a	NI_REMHLDREL	Remote hold released.
0x7b	NI_CALLFWDED	Call is forwarded.

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 92	ANSI 95	BICC	ETSI V2	ETSI V3
notifInd	*	*	*	*	*

Tokens for the ITU and Q.767 variants

Token	ITU White	ITU 97	Q.767
notifInd	*	*	*

Operator services information

Associated variant: ANSI 95

Contains information sent as an optional parameter in a call setup message.

```
typedef struct _opServInfo  /* Optional Backward Call Ind. */
{
    ElmtHdr eh;             /* element header             */
    TknU8   infoType;       /* information type           */
    TknU8   infoVal;        /* information value          */
} SiOpServInfo;
```

The fields in the SiOpServInfo structure are encoded as follows:

Field	Value
infoType	Information type. Defined values: 0x01 = OSITYP_ORIGACC Original access prefix 0x02 = OSITYP_BILLINFO Bill-to information entry and handling type 0x03 = OSITYP_BILLTYPE Bill-to type 0x05 = OSITYP_SPECHAND Special handling 0x07 = OSITYP_ACCSIG Access signaling
	Information value. Defined values depend on the value of infoType.

The following table shows the defined values for infoVal:

Information type	Defined values
OSITYP_ORIGACC	0x00 = OSIVAL_UNKNOWN Unknown 0x01 = OSIVAL_ONEPLUS 1+ or 011+ 0x03 = OSIVAL_ZERO 0-
OSITYP_BILLINFO	0x01 = OSIVAL_MANSTAT Information entry manual by operator, station handling 0x02 = OSIVAL_MANPERS Information entry manual by operator, person handling 0x03 = OSIVAL_AUTOTONESTAT Information entry automated by tone input, station handling 0x04 = OSIVAL_UNKSTAT Information entry unknown, station handling 0x05 = OSIVAL_UNKPERS Information entry unknown, person handling 0x06 = OSIVAL_MANUNK Information entry manual by operator, unknown handling 0x07 = OSIVAL_AUTOTONEUNK Information entry automated by tone input, unknown handling 0x08 = OSIVAL_AUTOTONEPERS Information entry automated by tone input, person handling 0x09 = OSIVAL_AUTOSPUNK Information entry automated by spoken input, unknown handling station handling 0x0b = OSIVAL_AUTOSPPERS Information entry automated by spoken input, person handling
OSITYP_BILLTYPE	0x01 = OSIVAL_CARD14 Card – 14 digit format 0x02 = OSIVAL_CARD89C Card – 89C format 0x03 = OSIVAL_CARDOTHER Card – other format 0x04 = OSIVAL_COLLECT Collect 0x05 = OSIVAL_THIRDNUM Third number 0x06 = OSIVAL_SENTPAID Sent paid

Information type	Defined values
OSITYP_BILLSPEC	0x01 = OSIVAL_NIDBAUTH NIDB authorizes 0x02 = OSIVAL_NIDBRPTAUTO NIDB reports, verify by automated means 0x03 = OSIVAL_NIDBRPTOPER NIDB reports, verify by operator 0x04 = OSIVAL_NONIDBQRY No NIDB query 0x05 = OSIVAL_NONIDBRSP No NIDB response 0x06 = OSIVAL_NIDBRPTUNAVAIL NIDB reports unavailable 0x07 = OSIVAL_NONIDBRSPTMOUT No NIDB response - timeout 0x08 = OSIVAL_NONIDBRSPREJ No NIDB response – reject component 0x09 = OSIVAL_NONIDBRSPACG No NIDB response – ACG in effect 0x0a = OSIVAL_NONIDBSCCPFAIL No NIDB response – SCCP failure
OSITYP_SPECHAND	0x01 = OSIVAL_CALLCOMP Call completion 0x02 = OSIVAL_RATEINFO Rate information 0x03 = OSIVAL_TROUBLE Trouble reporting 0x04 = OSIVAL_TIMECHRG Time and charges 0x05 = OSIVAL_CREDIT Credit reporting 0x06 = OSIVAL_ASSIST General assistance
OSITYP_ACCSIG	0x01 = OSIVAL_DIAL Dial pulse 0x02 = OSIVAL_DTMF Dual tone multifrequency

Tokens

Token	ANSI 95
infoType	*
infoVal	*

Optional backward call indicators

Associated variants: All except ANSI 88

Contains information sent in the backward direction to notify the originating exchange of additional information about a call in progress.

```
typedef struct _optBckCalInd    /* Optional Backward Call Ind.          */
{
    ElmtHdr eh;                 /* element header                       */
    TknU8   inbndInfoInd;       /* in-band information indicator        */
    TknU8   caFwdMayOcc;        /* call forward may occur ind.          */
    TknU8   segInd;             /* simple segmentation indicator        */
    TknU8   netDelay;           /* network excessive delay indicator    */
    TknU8   usrNetIneractInd;   /* user-network interaction ind.        */
    TknU8   mlppUsrInd;         /* MLPP user indicator                  */
    TknU8   spare;              /* spare bits                           */
    TknU8   reserved;           /* reserved bits                        */
} SiOptBckCalInd;
```

The fields in the SiOptBckCalInd structure are encoded as follows:

Field	Value
inbndInfoInd	In-band information indicator. Defined values: 0x00 = INBND_NOIND No indication 0x01 = INBND_AVAIL In-band information or an appropriate pattern is now available
caFwdMayOcc	Call forward may occur indicator. Defined values 0x00 = CAFWD_NOIND No indication 0x01 = CAFWD_MAYOCC Call diversion may occur
segInd	Simple segmentation indicator. Defined values: 0 = NOTSEGMENTED No indication will be sent 1 = SEGMENTED Additional information will be sent in a segmentation message
netDelay	Network excessive delay indicator. Defined values: 0 = No indication 1 = Delay encountered
usrNetIneractInd	User-network interaction indicator. Defined values: 0x00 = USERNET_NOIND No indication 0x01 = USERNET_INTERACTOCCUR User-network interaction occurs, cut through in both directions
mlppUsrInd	MLPP user indicator. Defined values: 0 = No indication 1 = MLPP user

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 92	ANSI 95	BICC	ETSI V2	ETSI V3
inbndInfoInd	*	*	*	*	*
caFwdMayOcc	*	*	*	*	*
segInd			*	*	*
netDelay		*			
usrNetIneractInd	*	*			
mlppUsrInd			*	*	*
spare	C-D	C-E			
reserved	E-G	E-F	E-H	E-H	E-H

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White	ITU 97	Q.767
inbndInfoInd	*	*	*	*
caFwdMayOcc	*	*	*	*
segInd		*	*	
netDelay				
usrNetIneractInd				
mlppUsrInd		*	*	
spare	C-D			C-D
reserved	E-H	E-H	E-H	

Optional forward call indicators

Associated variants: ANSI 88, BICC, ETSI V2, ETSI V3, ITU Blue, ITU White, ITU 97, Q.767

Contains information sent in the forward direction to notify the far exchange of additional information about a call in progress.

```
typedef struct _opFwdCalInd /* Optional Forward Call Indicators          */
{
    ElmtHdr eh;                /* element header                           */
    TknU8   clsdUGrpCaInd;     /* closed user group call ind.              */
    TknU8   segInd;            /* simple segmentation indicator            */
    TknU8   spare;             /* spare (4 bits)                           */
    TknU8   clidReqInd;        /* connected line identity request indicator */
    TknU8   ccbsCallInd;       /* CCBS call indicator                      */
    TknU8   callgPtyNumIncomInd; /* calling party number incomplete indicator */
    TknU8   connAddrReqInd1;   /* connected address request indicator      */
} SiOpFwdCalInd;
```

The fields in the SiOpFwdCalInd structure are encoded as follows:

Field	Value
clsdUGrpCaInd	Closed user group call indicator. Defined values for all supported variants except ANSI: 0x00 = CUG_NONCUG Non-CUG call 0x02 = CUG_ACCALLOW Closed user group call, outgoing access allowed 0x03 = CUG_ACCNOTALLOW Closed user group call, outgoing access not allowed
segInd	Simple segmentation indicator. Defined values: 0 = NOTSEGMENTED No indication will be sent 1 = SEGMENTED Additional information will be sent in a segmentation message
clidReqInd	Connected line identity request indicator. Defined values: 0x00 = CLIDNOTREQ Connected line identity not requested 0x01 = CLIDREQ Connected line identity requested
	CCBS call indicator. Defined values: 0x00 = NOTCCBSCALL Not a CCBS call 0x01 = CCBSCALL CCBS call
callgPtyNumIncomInd	Calling party number incomplete indicator. Defined values: 0x00 = CALLGPTYNUMCOMPL Calling party number complete 0x01 = CALLGPTYNUMINCOMPL Calling party number incomplete
connAddrReqInd1	Connected address request indicator. Defined values: 0x00 = CONNADDRNOTREQ Connected address not requested 0x01 = CONNADDRREQ Connected address requested

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	BICC	ETSI V2	ETSI V3
clsdUGrpCaInd	*	*	*	*
segInd		*	*	*
spare	C-D	D-G	D-G	D-G
clidReqInd		*	*	*
ccbsCallInd	*			
callgPtyNumIncomInd	*			
connAddrReqInd1	*			

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White		Q.767
clsdUGrpCaInd	*	*	*	*
segInd		*	*	
spare	C-H	D-G	D-G	C-G
clidReqInd		*	*	*
ccbsCallInd				
callgPtyNumIncomInd				
connAddrReqInd1				

Original called number

Associated variants: ANSI 92, ANSI 95, BICC, ETSI V2, ETSI V3, ITU Blue, ITU White, ITU 97

Identifies the address of the party that initiates the redirection when call redirecting (forwarding) occurs.

```
typedef struct _origCdNum  /* Original Called Number            */
{
  ElmtHdr eh;              /* element header                    */
  TknU8   natAddr;         /* nature of address indicator       */
  TknU8   oddEven;         /* odd or even                       */
  TknU8   spare1;          /* spare bits                        */
  TknU8   presRest;        /* Presentation restricted ind.      */
  TknU8   numPlan;         /* numbering plan                    */
  TknU8   spare2;          /* spare bits                        */
  TknStr  addrSig;         /* Address Signal                    */
} SiOrigCdNum;
```

The fields in the SiOrigCdNum structure are encoded as follows:

Field	Value
natAddr	Nature of address indicator. Defined values for all variants: 0x01 = SUBSNUM Subscriber number 0x03 = NATNUM Nationally significant number 0x04 = INTNATNUM International number Additional values for ANSI: 0x71 = SUBSNUMOPREQ Subscriber number operator requested 0x72 = NATNUMOPREQ National number operator requested 0x73 = INTNATNUMOPREQ International number operator requested 0x74 = NONUMPRESOPREQ No number present operator requested 0x75 = NONUMPRESCUTTHRU No number present cut-through call to carrier 0x76 = NINEFIVEOH 950+ service 0x77 = TSTLINETSTCODE Test line test code
oddEven	Specifies whether the number of digits in the addrSig field is even or odd. If even, the last octet contains two digits. If odd, the last octet contains only one digit and the most significant four bits not used. Defined values: 0 = NMB_EVEN 1 = NMB_ODD
presRest	Address presentation restricted indicator. Defined values: 0x00 = PRESALLOW Presentation allowed 0x01 = PRESREST Presentation restricted
numPlan	Numbering plan. Defined values for all supported variants except BICC: 0x00 = NP_UNK Unknown 0x01 = NP_ISDN ISDN/telephony according to E.164/E.163 0x02 = NP_TEL Telephony numbering according to E.163; spare in ITU Blue, ITU White, and ITU 97 0x03 = NP_DATA Data numbering according to X.121 0x04 = NP_TELEX Telex number according to F.69 0x08 = NP_NATIONAL National standard numbering 0x09 = NP_PRIVATE Private numbering plan 0x0f = NP_EXT Reserved for extension Defined values for BICC: 0x00 = NP_UNK Unknown 0x01 = NP_ISDN ISDN/telephony according to ITU-T E.164/E.163 0x02 = NP_TEL Spare 0x03 = NP_DATA Data numbering according to ITU-T X.121 0x04 = NP_TELEX Telex number according to ITU_T F.69 0x05 = NP_PRIVATE Private numbering plan. 0x06 = NP_NATIONAL Reserved for national use
addrSig	Actual address digits, encoded as shown in the following tables.

For the addrSig field, the actual address digits are encoded as follows:

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

where each digit is encoded with the following bit pattern:

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

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 92	ANSI 95	BICC	ETSI V2	ETSI V3
natAddr	*	*	*	*	*
oddEven	*	*	*	*	*
spare1	2(1-2)	2(1-2)		2(1-2)	2(1-2)
presRest	*	*	*	*	*
numPlan	*	*	*	*	*
spare2	2(8)	2(8)	2(8)	2(8)	2(8)
addrSig	*	*	*	*	*

Tokens for the ITU variants

Token	ITU Blue	ITU White	ITU 97
natAddr	*	*	*
oddEven	*	*	*
spare1	2(1-2)	2(1-2)	2(1-2)
presRest	*	*	*
numPlan	*	*	*
spare2	2(8)	2(8)	2(8)
addrSig	*	*	*

Originating line information

Associated variants: ANSI 88 ANSI 92, ANSI 95

Passes originating line information to the far exchange in the initial address message in ANSI networks.

```
typedef struct _origLineInf   /* Originating Line Info */
{
    ElmtHdr eh;               /* element header        */
    TknU8   lineInfo;         /* originating line info */
} SiOrigLineInf;
```

The lineInfo field is encoded with one of the following values:

0x00	OL_IDENTLINE	
0x01	OL_ONI	ONI (multiparty).
0x02	OL_ANIFAIL	ANI failure (unavailable).
0x05	OL_HOTEL	
0x07		Coinless, hospital, inmate.
0x08	OL_INTERLATA	interLATA restricted.
0x14	OL_AIOD	
0x1b	OL_COINELINE	Coin line.
0x44	OL_INTERLATA_HOTEL	interLATA restricted-hotel.
0x4e	OL_INTERLATA_COINLESS	interLATA restricted-coinless.

Tokens

Token	ANSI 88	ANSI 92	ANSI 95
lineInfo	*	*	*

Outgoing trunk group number

Associated variants: ANSI 88, ANSI 92, ANSI 95

Provides the trunk group number used for an interworking call.

```
typedef struct _outgTrkGrpNum  /* Outgoing Trunk Grp Number */
{
    ElmtHdr eh;                /* element header            */
    TknStr  digits;            /* digits                    */
} SiOutgTrkGrpNum;
```

The encoding of the digits field is implementation specific.

Tokens

Token	ANSI 88	ANSI 92	ANSI 95
digits	*	*	*

Parameter compatibility

Associated variants: BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Contains information sent in either direction to instruct the far exchange how to treat unrecognized parameters.

```
typedef struct _parmCom    /* parameter compatibility information */
{
    ElmtHdr eh;            /* element header                      */
    TknU8   upgrPar1;      /* upgraded parm 1                     */
    TknU8   tranXInd1;     /* transit exchange indicator          */
    TknU8   relCllInd1;    /* release call indicator              */
    TknU8   sndNotInd1;    /* send notification indicator         */
    TknU8   dcrdMsgInd1;   /* discard message indicator           */
    TknU8   spare1;                                               */
    TknU8   dcrdParInd1;   /* discard parameter indicator         */
    TknU8   upgrPar2;      /* upgraded parm 2                     */
    TknU8   tranXInd2;     /* transit exchange indicator          */
    TknU8   relCllInd2;    /* release call indicator              */
    TknU8   sndNotInd2;    /* send notification indicator         */
    TknU8   dcrdMsgInd2;   /* discard message indicator           */
    TknU8   dcrdParInd2;   /* discard parameter indicator         */
    TknU8   spare2;                                               */
    TknU8   upgrPar3;      /* upgraded parm 3                     */
    TknU8   tranXInd3;     /* transit exchange indicator          */
    TknU8   relCllInd3;    /* release call indicator              */
    TknU8   sndNotInd3;    /* send notification indicator         */
    TknU8   dcrdMsgInd3;   /* discard message indicator           */
    TknU8   dcrdParInd3;   /* discard parameter indicator         */
    TknU8   spare3;                                               */
} SiParmCompInfo;
```

For the BICC variant, the following structure is used for sending information regarding unrecognized parameters. The application should not use both structures at the same time.

```
typedef struct _biccParmCom   /* parameter compatibility information for ANSI and ITU
BICC */
{
   ElmtHdr eh;                    /* element header */
   TknU8   upgrPar1;              /* upgraded parm 1 */
   TknU8   tranXInd1;             /* transit exchange indicator */
   TknU8   relCllInd1;            /* release call indicator */
   TknU8   sndNotInd1;            /* send notification indicator */
   TknU8   dcrdMsgInd1;           /* discard message indicator */
   TknU8   dcrdParInd1;           /* discard parameter indicator */
   TknU8   passNotPoss1;          /* pass on not possible indicator */
   TknU8   bnIntwInd1;            /* broadband/narrowband interworking indicator */
   TknU8   spare1;                /* spare */
   TknU8   upgrPar2;              /* upgraded parm 2 */
   TknU8   tranXInd2;             /* transit exchange indicator */
   TknU8   relCllInd2;            /* release call indicator */
   TknU8   sndNotInd2;            /* send notification indicator */
   TknU8   dcrdMsgInd2;           /* discard message indicator */
   TknU8   dcrdParInd2;           /* discard parameter indicator */
   TknU8   passNotPoss2;          /* pass on not possible indicator */
   TknU8   bnIntwInd2;            /* broadband/narrowband interworking indicator */
   TknU8   spare2;                /* spare */
    TknU8   upgrPar3;              /* upgraded parm 3 */
   TknU8   tranXInd3;             /* transit exchange indicator */
   TknU8   relCllInd3;            /* release call indicator */
   TknU8   sndNotInd3;            /* send notification indicator */
   TknU8   dcrdMsgInd3;           /* discard message indicator */
   TknU8   dcrdParInd3;           /* discard parameter indicator */
   TknU8   passNotPoss3;          /* pass on not possible indicator */
   TknU8   bnIntwInd3;            /* broadband/narrowband interworking indicator */
   TknU8   spare3;                /* spare */
} SiBiccParmCompInfo;
```

The fields in the SiParmCompInfo structure are encoded as follows:

Field	Value
upgrPar***n***	Parameter name code for parameter ***n*** as specified in ITU-T Recommendation Q.763 (Table 5). You can specify information for up to three parameters.
tranXInd***n***	Transit exchange indicator. Defined values: 0x00 = Transit exchange interpretation 0x01 = End node interpretation
relCllInd***n***	Release call indicator. Defined values: 0x00 = Do not release call 0x01 = Release call
sndNotInd***n***	Send notification indicator. Defined values: 0x00 = Do not send notification 0x01 = Send notification (confusion or release complete message)
dcrdMsgInd***n***	Discard message indicator. Defined values: 0x00 = Do not discard (pass on) message
dcrdParInd***n***	0x00 = Do not discard (pass on) parameter 0x01 = Discard parameter
passNotPoss***n*** (used for BICC variant)	Pass on not possible indicator. Defined values: 0x00 = Release call 0x01 = Discard message 0x02 = Discard parameter 0x03 = Reserved
bnIntwInd***n*** (used for BICC variant)	Broadband/narrow-band interworking indicator. Defined values: 0x00 = Pass on 0x01 = Discard message 0x02 = Release call 0x03 = Discard parameter

Tokens

Token	BICC	ETSI V2	ETSI V3	ITU White	ITU 97
upgrPar1	*	*	*	*	*
tranXInd1	*	*	*	*	*
relCllInd1	*	*	*	*	*
sndNotInd1	*	*	*	*	*
dcrdMsgInd1	*	*	*	*	*
spare1	*	*	*	*	*
dcrdParInd1	*	*	*	*	*
passNotPoss1	*				
bnIntwInd1	*				
upgrPar2	*	*	*	*	*
tranXInd2	*	*	*	*	*

Token	BICC	ETSI V2	ETSI V3	ITU White	ITU 97
relCllInd2	*	*	*	*	*
sndNotInd2	*	*	*	*	*
dcrdMsgInd2	*	*	*	*	*
dcrdParInd2	*	*	*	*	*
passNotPoss2	*				
bnIntwInd2	*				
spare2	*	*	*	*	*
upgrPar3	*	*	*	*	*
tranXInd3	*	*	*	*	*
relCllInd3	*	*	*	*	*
sndNotInd3	*	*	*	*	*
dcrdMsgInd3	*	*	*	*	*
dcrdParInd3	*	*	*	*	*
passNotPoss3	*				
bnIntwInd3	*				
spare3	*	*	*	*	*

Pass along

Associated variants: ANSI 92, ANSI 95, ETSI V2, ETSI V3, ITU White, ITU 97

Passes any message between two signal points along the same signaling path used to set up a connection between those two points in a pass-along message.

```
typedef struct _passAlng  /* Pass Along     */
{
    ElmtHdr eh;           /* element header */
    TknStr   passAlng;    /* pass along     */
} SiPassAlng;
```

The passAlng field contains binary data passed transparently between signal points.

Tokens

Token	ANSI 92	ANSI 95	ETSI V2	ETSI V3	ITU White	ITU 97
passAlng	*	*	*	*	*	*

Propagation delay

Associated variants: BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Contains information sent in the forward direction to indicate the propagation delay of a connection.

```
typedef struct _propDly   /* propagation delay */
{
    ElmtHdr eh;           /* element header    */
    TknU16  delayVal;     /* delay value       */
} SiPropDly;
```

The delayVal field is the propagation delay in milliseconds.

Tokens

Token	BICC	ETSI V2	ETSI V3	ITU White	ITU 97
delayVal	*	*	*	*	*

Range and status

Associated variants: All

Identifies a range of circuits and a status that affects that range in circuit supervision messages.

```
typedef struct _rangStat  /* Range and Status */
{
    ElmtHdr eh;           /* element header   */
    TknU8   range;        /* range            */
    TknStr  status;       /* status           */
} SiRangStat;
```

The fields in the SiRangStat structure are encoded as follows:

Field	Value
range	Binary number from 1 - 255 that identifies the range of circuits/CICs.
status	Array of status bits with a single bit status field for each circuit in the specified range. The first circuit status bit is in the least significant bit of the first octet. The meaning of the status bit depends on the message that it is in. Refer to the relevant variant recommendation for status bit encoding rules.

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	ANSI 92	ANSI 95		ETSI V2	ETSI V3
range	*	*	*	*	*	*
status	*	*	*	*	*	*

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White	ITU 97	
range	*	*	*	*
status	*	*	*	*

Redirect capability

Associated variants: BICC, ITU 97

Indicates that at least one node in the connection can redirect the call. Redirect capability is an extended element. Refer to *Extended element* on page 185 for more information.

Redirect counter

Associated variants: BICC, ITU 97

Indicates the number of times a call has been redirected. Redirect counter is an extended element. Refer to *Extended element* on page 185 for more information.

Redirecting number

Associated variants: All except Q.767

Provides the redirecting number format.

```
typedef struct _redirgNum  /* Redirection Number           */
{
    ElmtHdr eh;            /* element header               */
    TknU8   natAddr;       /* nature of address indicator  */
    TknU8   oddEven;       /* odd or even                  */
    TknU8   spare1;        /* spare bits                   */
    TknU8   presRest;      /* Presentation restricted ind. */
    TknU8   numPlan;       /* numbering plan               */
    TknU8   spare2;        /* spare bits                   */
    TknStr  addrSig;       /* Address Signal               */
} SiRedirgNum;
```

The fields in the SiRedirgNum structure are encoded as follows:

Field	Value
natAddr	Nature of address indicator. Defined values for all variants: 0x01 = SUBSNUM Subscriber number 0x03 = NATNUM Nationally significant number 0x04 = INTNATNUM International number Additional values for ANSI: 0x71 = SUBSNUMOPREQ Subscriber number, operator requested 0x72 = NATNUMOPREQ National number, operator requested 0x73 = INTNATNUMOPREQ International number, operator requested 0x74 = NONUMPRESOPREQ No number present, operator requested 0x75 = NONUMPRESCUTTHRU No number present, cut-through call to carrier 0x76 = NINEFIVEOH 950+ service 0x77 = TSTLINETSTCODE Test line test code
oddEven	Specifies whether the number of digits in the addrSig field is even or odd. If even, the last octet contains two digits. If odd, the last octet contains only one digit and the most significant four bits not used. Defined values: 0 = NMB_EVEN 1 = NMB_ODD
presRest	Presentation restricted indicator. Defined values: 0x00 = PRESALLOW Presentation allowed 0x01 = PRESREST Presentation restricted 0x02 = ADDRNOAVAIL Address not available

Field	Value
numPlan	Numbering plan. Defined values for all supported variants except BICC: 0x00 = NP_UNK Unknown 0x01 = NP_ISDN ISDN/telephony according to ITU-T E.164/E.163 0x02 = NP_TEL Telephony numbering according to E.163; spare in ITU Blue, ITU White, and ITU 97 0x03 = NP_DATA Data numbering according to X.121 0x04 = NP_TELEX Telex number according to F.69 0x08 = NP_NATIONAL National standard numbering 0x09 = NP_PRIVATE Private numbering plan 0x0f = NP_EXT Reserved for extension Defined values for BICC: 0x00 = NP_UNK Unknown 0x01 = NP_ISDN ISDN/telephony according to ITU-T E.164/E.163 0x02 = NP_TEL Spare 0x03 = NP_DATA Data numbering according to ITU-T X.121 0x04 = NP_TELEX Telex number according to ITU_T F.69 0x05 = NP_PRIVATE Private numbering plan. 0x06 = NP_NATIONAL Reserved for national use
addrSig	Actual address digits, encoded as shown in the following tables.

For the addrSig field, the actual address digits are encoded as follows:

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

where each digit is encoded with the following bit pattern:

Bit pattern	Digit/signal
0000	0
0001	1
0010	2
0011	3
	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

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	ANSI 92	ANSI 95	BICC	ETSI V2	ETSI V3
natAddr	*	*	*	*	*	*
oddEven	*	*	*	*	*	*
spare1	2(1-2)	2(1-2)	2(1-2)	2(1-2)	2(1-2)	2(1-2)
presRest	*	*	*	*	*	*
numPlan	*			*	*	*
spare2	2(8)	2(8)	2(8)	2(8)	2(8)	2(8)
addrSig	*	*	*	*	*	*

Tokens for the ITU variants

Token	ITU Blue	ITU White	ITU 97
natAddr	*	*	*
oddEven	*	*	*
spare1	2(1-2)	2(1-2)	2(1-2)
presRest	*	*	*
numPlan	*	*	*
spare2	2(8)	2(8)	2(8)
addrSig	*	*	*

Redirection information

Associated variants: All except Q.767

Provides the redirection information format.

```
typedef struct _redirInfo   /* Redirection Information     */
{
    ElmtHdr eh;             /* element header              */
    TknU8   redirInd;       /* redirection indicator       */
    TknU8   spare1;         /* spare bits                  */
    TknU8   origRedirReas;  /* original redirection reason */
    TknU8   redirCnt;       /* redirection count           */
    TknU8   spare2;         /* spare bits                  */
    TknU8   redirReas;      /* redirection reason          */
} SiRedirInfo;
```

The fields in the redirInfo structure are encoded as follows:

Field	Value
redirInd	Redirection indicator. Defined values: 0x00 = RI_NOREDIR No redirection (national use) 0x01 = RI_CALLRERTE Call rerouted (national use) 0x02 = RI_CALLRERTEALLRIPRESRES Call rerouted, all redirection information presentation restricted (national use) 0x03 = RI_CALLFWD Call diverted 0x04 = RI_CALLFWDALLRIPRESRES Call diverted, all redirection information presentation restricted 0x05 = RI_CARERTEPRESRES Call rerouted, redirection number presentation restricted (national use) 0x06 = RI_CALLFWDPRESRES Call diversion, redirection number presentation restricted (national use)
origRedirReas	Original redirection reason. Defined values for all variants except ANSI: 0x00 = REAS_UNKNWN Unknown/not available 0x01 = REAS_USRBUSY User busy (national use) 0x02 = REAS_NOREPLY No reply (national use) 0x03 = REAS_UNCOND Unconditional (national use) 0x04 = REAS_DFLCDURALRT Deflection during alerting 0x05 = REAS_DFLCIMMDRSP Deflection immediate response 0x06 = REAS_MBLSUBNOTRCHBL Mobile subscriber not reachable Defined values for ANSI: 0x03 = REAS_FIXED 0x04 = REAS_VARIABLE
redirCnt	[1..5] for ITU-T [0..15] for ANSI
	Redirection reason. Defined values for all supported variants except ANSI: 0x00 = REAS_UNKNWN Unknown/not available 0x02 = REAS_NOREPLY No reply 0x03 = REAS_UNCOND Unconditional 0x04 = REAS_DFLCDURALRT Deflection during alerting 0x05 = REAS_DFLCIMMDRSP Deflection immediate response Defined values for ANSI: 0x03 = REAS_FIXED 0x04 = REAS_VARIABLE

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	ANSI 92	ANSI 95	BICC	ETSI V2	ETSI V3
redirInd	*	*	*	*	*	*
spare1	D			D	D	D
origRedirReas		*	*	*	*	*
redirCnt		*	*	*	*	*
spare2		1(4)	1(4)	L	L	L
redirReas	*	*	*	*	*	*

Tokens for the ITU variants

Token	ITU Blue		ITU 97
redirInd	*	*	*
spare1	D	D	D
origRedirReas	*	*	*
redirCnt	*	*	*
spare2	L	L	L
redirReas	*	*	*

Redirection number

Associated variants: ANSI 88, BICC, ETSI V2, ETSI V3, ITU Blue, ITU White, ITU 97

Provides the redirection number format.

```
typedef struct _redirNum   /* Redirection Number                  */
{
    ElmtHdr eh;            /* element header                      */
    TknU8   natAddr;       /* nature of addresss indicator        */
    TknU8   oddEven;       /* odd or even                         */
    TknU8   spare;         /* spare bits                          */
    TknU8   numPlan;       /* numbering plan                      */
    TknU8   innInd;        /* internal network number indicator   */
    TknStr  addrSig;       /* Address Signal                      */
} SiRedirNum;
```

The fields in the SiRedirNum structure are encoded as follows:

Field	Value
natAddr	Nature of address indicator. Defined values for all variants: 0x01 = SUBSNUM Subscriber number 0x03 = NATNUM Nationally significant number 0x04 = INTNATNUM International number Additional values for ANSI: 0x71 = SUBSNUMOPREQ Subscriber number, operator requested 0x72 = NATNUMOPREQ National number, operator requested 0x73 = INTNATNUMOPREQ International number, operator requested 0x74 = NONUMPRESOPREQ No number present, operator requested 0x75 = NONUMPRESCUTTHRU No number present, cut-through call to carrier 0x76 = NINEFIVEOH 950+ service 0x77 = TSTLINETSTCODE Test line test code
oddEven	Specifies whether the number of digits in the addrSig field is even or odd. If even, the last octet contains two digits. If odd, the last octet contains only one digit and the most significant four bits not used. Defined values: 0 = NMB_EVEN 1 = NMB_ODD
spare	Spare bits.
numPlan	Numbering plan. Defined values for all supported variants except BICC: 0x00 = NP_UNK Unknown 0x01 = NP_ISDN ISDN/telephony according to E.164/E.163 0x02 = NP_TEL Telephony numbering according to E.163; spare in ITU Blue, ITU White, and ITU 97 0x03 = NP_DATA Data numbering according to X.121 0x04 = NP_TELEX Telex number according to F.69 0x08 = NP_NATIONAL National standard numbering 0x09 = NP_PRIVATE Private numbering plan 0x0f = NP_EXT Reserved for extension Defined values for BICC: 0x00 = NP_UNK Unknown 0x01 = NP_ISDN ISDN/telephony according to ITU-T E.164/E.163 0x02 = NP_TEL Spare 0x03 = NP_DATA Data numbering according to ITU-T X.121 0x04 = NP_TELEX Telex number according to ITU_T F.69 0x05 = NP_PRIVATE Private numbering plan. 0x06 = NP_NATIONAL Reserved for national use
innInd	Internal network number indicator. Defined values: 1 = INN_NOTALLOW
addrSig	Actual address digits, encoded as shown in the following tables.

For the addrSig field, the actual address digits are encoded as follows:

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

where each digit is encoded with the following bit pattern:

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

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	BICC	ETSI V2	ETSI V3
natAddr	*	*	*	*
oddEven	*	*	*	*
spare	2(1-2)	2(1-2)	2(1-2)	2(1-2)
numPlan	*	*	*	*
innInd	*	*	*	*
addrSig	*	*	*	*

Tokens for the ITU variants

Token	ITU Blue	ITU White	ITU 97
natAddr	*	*	*
oddEven	*	*	*
spare	2(1-2)	2(1-2)	2(1-2)
numPlan	*	*	*
innInd	*	*	*
addrSig	*	*	*

Redirection restriction

Associated variant: BICC, ITU White

Contains information sent in the backward direction to indicate whether the diverted-to user supports presentation of the number.

```
typedef struct _redirRstr  /* redirection restriction   */
{
    ElmtHdr eh;            /* element header            */
    TknU8   presRest;      /* presentation restriction  */
    TknU8   spare;         /* spare bits                */
} SiRedirRestr;
```

The presRest field is encoded as follows:

0x00	PRESALLOW	Presentation allowed.
0x01	PRESREST	Presentation restricted.

Tokens

Token	BICC	ITU White
presRest	*	*
spare	C-H	C-H

Remote operations

Associated variants: ANSI 95, BICC, ITU White

Invokes a supplementary service identified by an operation value and carries the results of the operation.

```
typedef struct __remotOper   /* remote operations     */
{
    ElmtHdr eh;              /* element header        */
    TknU8   protProf;        /* protocol profile      */
    TknU8   spare;           /* spare bits            */
    TknStr  compon;          /* components            */
} SiRemotOper;
```

The protProf field is encoded as follows:

0x11	PP_REMOPPROT	Remote operations protocol.

For information about the encoding of the compon (components) field, refer to ANSI T1[1].113 or to ITU-T Recommendation Q.763.

Tokens

Token	ANSI 95	BICC	ITU White
protProf	*	*	*
spare	1(6-7)	1(6-7)	1(6-7)
compon	*	*	*

SCF ID

Associated variants: BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Provides the SCF ID format.

```
typedef struct_scfID     /* SCF ID         */
{
    ElmtHdr eh;           /* element header */
    TknStr  data;         /* status         */
}SiScfID;
```

Tokens

Token	BICC	ETSI V2	ETSI V3	ITU White	ITU 97
data	*	*	*	*	*

Service activation

Associated variants: ANSI 92, ANSI 95, BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Invokes supplementary services from another exchange.

```
typedef struct _serviceAct  /* Service Activation     */
{
    ElmtHdr eh;             /* element header         */
    TknStr  serviceAct;     /* service activation     */
} SiServiceAct;
```

The serviceAct field is an array of network implementation-specific service codes.

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 92	ANSI 95	BICC	ETSI V2	ETSI V3
serviceAct	*	*	*	*	*

Tokens for the ITU variants

Token	ITU White	ITU 97
serviceAct	*	*

Service code ID

Associated variants: ANSI 92, ANSI 95

Provides the service code ID format.

```
typedef struct _serviceCode   /* Service Code    */
{
    ElmtHdr eh;               /* element header  */
    TknU8   serviceCode;      /* service code    */
} SiServiceAct;
```

Tokens

Token	ANSI 92	ANSI 95
serviceCode	*	*

Signaling point code

Associated variants: ANSI 88, ETSI V2, ETSI V3, ITU Blue, ITU White, ITU 97

Identifies the signaling point at which a call failed in a release message.

```
typedef struct _sigPointCode  /* Signaling Point Code */
{
    ElmtHdr eh;               /* element header       */
    TknU32  sigPointCode;     /* signaling point code */
} SiSigPointCode;
```

The sigPointCode field is encoded as a 32-bit quantity of which the least significant 24 bits (ANSI 88) or the least significant 14 bits (ETSI and ITU) are used. For example, an ANSI point code represented by the decimal string 1.4.7 is encoded as the hexadecimal number 0x00010407.

Tokens

Token	ANSI 88	ETSI V2	ETSIV3	ITU Blue	ITU White	ITU 97
sigPointCode	*	*		*	*	*

Special processing request

Associated variants: ANSI 92, ANSI 95

Indicates the special requirements needed for a connection originating from a private network and going to a public network.

```
typedef struct _specProcReq  /* Special Processing Request */
{
    ElmtHdr eh;              /* element header             */
    TknU8   specProcReq;     /* special processing request */
} SiSpecProcReq;
```

The only defined value for the specProcReq field is 0x7f, service processing required.

Tokens

Token	ANSI 92	ANSI 95
specProcReq	*	*

Subsequent number

Associated variants: BICC, ETSI V2, ETSI V3, ITU Blue, ITU White, ITU 97, Q.767

Conveys additional called party address information to the far exchange in a subsequent address message.

```
typedef struct _subNum     /* Subsequent Number */
{
    ElmtHdr eh;            /* element header    */
    TknU8   oddEven;       /* odd or even       */
    TknU8   spare;         /* spare bits        */
    TknStr  addrSig;       /* Address Signal    */
} SiSubNum;
```

The fields in the SiSubNum structure are encoded as follows:

Field	Value
oddEven	Specifies whether the number of digits in the addrSig field is even or odd. If even, the last octet contains two digits. If odd, the last octet contains only one digit and the most significant four bits not used. Defined values: 0 = NMB_EVEN 1 = NMB_ODD
addrSig	Actual address digits, encoded as shown in the following tables.

For the addrSig field, the actual address digits are encoded as follows:

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

where each digit is encoded with the following bit pattern:

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

Tokens for the BICC, and ETSI variants

Token	BICC	ETSI V2	ETSI V3
oddEven	*	*	*
spare	1(1-7)	1(1-7)	1(1-7)
addrSig		*	*

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White	ITU 97	Q.767
oddEven	*	*	*	*
spare	1(1-7)	1(1-7)	1(1-7)	1(1-7)
addrSig	*	*	*	*

Suspend or resume indicators

Associated variants: All

Indicates whether the suspend or resume message was initiated by an ISDN subscriber or by the network.

```
typedef struct _susResInd  /* Suspend/Resume indicators */
{
    ElmtHdr eh;            /* element header            */
    TknU8   susResInd;     /* suspend/resume indicators */
    TknU8   spare;         /* spare bits                */
} SiSusResInd;
```

The susResInd field is encoded to one of the following values:

0x00	SR_ISDNSUBINIT	ISDN subscriber initiated.
0x01	SR_NETINIT	Network initiated.

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	ANSI 92	ANSI 95	BICC		ETSI V3
susResInd	*	*	*	*	*	*
spare	B-H	B-H	B-H	B-H	B-H	B-H

Tokens for the ITU and Q.767 variants

Token	ITU Blue		ITU 97	Q.767
susResInd	*	*	*	*
spare	B-H	B-H	B-H	B-H

Transaction request

Associated variants: ANSI 92, ANSI 95

Enables the ISUP protocol to use the TCAP layer to deliver service information related to a call. Refer to the ANSI TCAP recommendations for information about the transaction request information element.

```
typedef struct _transReq   /* Transaction Request */
{
    ElmtHdr eh;            /* element header      */
    TknU32  transId;       /* transaction id      */
    TknStr  SCCPAddr;      /* SCCP address        */} SiTransReq;
```

Tokens

Token	ANSI 92	ANSI 95
transId	*	*
SCCPAddr	*	*

Transit network selection

Associated variants: All except Q.767

Identifies the transit network used to carry a call.

```
typedef struct _tranNetSel   /* Transit Network Selection */
{
    ElmtHdr eh;              /* element header            */
    TknU8   netIdPln;        /* network id plan           */
    TknU8   typNetId;        /* type of network id        */
    TknU8   oddEven;         /* odd/even                  */
    TknU8   spare;           /* spare bits                */
    TknStr  netId;           /* network identification    */
} SiTranNetSel;
```

The fields in the SiTranNetSel structure are encoded as follows:

Field	Value
netIdPln	Network ID plan. Defined values for all supported variants except ANSI 92: 0x00 = NI_UNKNWN 0x03 = NI_DNIC_X21 0x06 = NI_MNIC_E212 Additional values for ANSI 92: 0x01 = NI_3DIGCIC 0x02 = NI_4DIGCIC
typNetId	Type of network ID. Defined values: 0x00 = TNI_CCITT 0x02 = TNI_NATNET
oddEven	Specifies whether the number of digits in the addrSig field is even or odd. If even, the last octet contains two digits. If odd, the last octet contains only one digit and the most significant four bits not used. Defined values: 0 = NMB_EVEN 1 = NMB_ODD
netId	An implementation-specific network identifier. For example, Bellcore networks may use the same three digit code as when dialing 10xxx to identify an interexchange carrier. The value is encoded as shown in the following tables.

For the addrSig field, the actual address digits are encoded as follows:

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

where each digit is encoded with the following bit pattern:

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

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	ANSI 92		BICC	ETSI V2	ETSI V3
netIdPln	*	*	*	*	*	*
typNetId	*	*	*	*	*	*
oddEven				*	*	*
spare	1(8)	1(8)	1(8)			
netId	*	*	*	*	*	*

Tokens for the ITU variants

Token	ITU Blue	ITU White	ITU 97
netIdPln	*	*	*
typNetId	*	*	*
oddEven	*	*	*
spare			
netId	*	*	*

Transmission medium requirement

Associated variants: BICC, ETSI V2, ETSI V3, ITU Blue, ITU White, ITU 97, Q.767

Contains information sent in the forward direction to indicate the type of transmission medium required for the connection.

```
typedef struct _txMedReq /* Transmission Medium Requirement */
{
    ElmtHdr eh;          /* element header                  */
    TknU8   trMedReq;    /* transmission medium requirement */
} SiTxMedReq;
```

The trMedReq field is encoded to one of the following values:

0x00	TMR_SPEECH	Speech.
0x02	64 kbit/s unrestricted	64 kbit/s unrestricted.
0x03	TMR_31KHZ	3.1 kHz audio.
0x05	TMR_ALTSPEECH	Reserved for alternate speech (service 2)/64 kbit/s unrestricted (service 1).
0x05	TMR_ALT64KBITS	Reserved for alternate 64 kbit/s unrestricted (service 1) speech (service 2).
0x08	TMR_2X64KBITS	TMR_2X64KBITS 2 x 64 kbit/s unrestricted.
0x09	TMR_1536KBITS	1536 kbit/s unrestricted.
0x0a	TMR_1920KBITS	1920 kbit/s unrestricted.

Tokens for the BICC, and ETSI variants

Token	BICC	ETSI V2	ETSI V3
trMedReq	*	*	*

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White	ITU 97	Q.767
trMedReq	*	*	*	*

UID action indicators

Associated variants: BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Provides the UID action indicators format.

```
typedef struct _uidActionInd  /* UID action indicators                     */
{
    ElmtHdr eh;               /* element header                            */
    TknU8   thruConnInd;      /* Through-connect instruction indicator     */
    TknU8   t9Ind;            /* T9 indicator                              */
    TknU8   spare1;           /* bits C-G                                  */
}SiUIDActionInd;
```

The fields in the SiUIDActionInd structure are encoded as follows:

Field	Value
thruConnInd	Through-connect instruction indicator. Defined values: 0 = THRU_CONN_NO_INDICATION No indication 1 = THRU_CONN_BOTH_DIRS Through-connection in both directions
t9Ind	T9 indicator. Defined values: 0 = T9_NO_INDICATION No indication 1 = T9_STOP Stop or do not start T9 timer

Tokens

Token	BICC	ETSI V2	ETSI V3	ITU White	ITU 97
thruConnInd	*	*	*	*	*
t9Ind	*	*	*	*	*
spare1	C-G	C-G	C-G	C-G	C-G

UID capability indicators

Associated variants: BICC, ETSI V2, ETSI V3, ITU White, ITU 97

Provides the UID capability indicators format.

```
typedef struct _uidCapInd    /* UID Capability Indicators              */
{
    ElmtHdr eh;              /* element header                         */
    TknU8   thruConnInd;     /* Through-connect instruction indicator */
    TknU8   t9Ind;           /* T9 Indicator                           */
    TknU8   spare1;          /* bits C-G                               */
}SiUIDConnInd;
```

The fields in the SiUIDConnInd structure are encoded as follows:

Field	Value
thruConnInd	Through-connect instruction indicator. Defined values: 0 = THRU_CONN_IND_NO_INDICATION 1 = THRU_CONN_IND_MOD_POSSIBLE
t9Ind	T9 indicator. Defined values: 0 = T9_NO_INDICATION 1 = T9_STOPPING_POSSIBLE

Tokens

Token	BICC	ETSI V2	ETSI V3	ITU White	ITU 97
thruConnInd	*	*	*	*	*
t9Ind	*	*	*	*	*
spare1	C-G	C-G	C-G	C-G	C-G

User service information

Associated variants: All

Indicates the bearer capability requested by the calling party.

```
typedef struct _usrServInfo /* User Service Information                */
{
    ElmtHdr eh;             /* element header                          */
    TknU8   infoTranCap;    /* info transfer capability                */
    TknU8   cdeStand;       /* coding standard                         */
    TknU8   infoTranRate0;  /* information transfer rate               */
    TknU8   tranMode;       /* transfer mode                           */
    TknU8   establish;      /* establishment                           */
    TknU8   config;         /* configuration                           */
    TknU8   chanStruct;     /* structure                               */
    TknU8   infoTranRate1;  /* information transfer rate               */
    TknU8   symmetry;       /* symmetry                                */
    TknU8   usrInfLyr1Prot; /* user info layer 1 protocol              */
    TknU8   lyr1Ident;      /* layer 1 identity                        */
    TknU8   usrRate;        /* user rate                               */
    TknU8   negot;          /* negotiation                             */
    TknU8   syncAsync;      /* synchronous/asynchronous                */
    TknU8   flcOnRx;        /* flow control on reception               */
    TknU8   flcOnTx;        /* flow control on transmission            */
    TknU8   niClkOnRx;      /* network independent clock on reception  */
    TknU8   niClkOnTx;      /* network independent clock on transmission */
    TknU8   interRate;      /* intermediate rate                       */
    TknU8   inOutBandNeg;   /* inband/outband negotiation              */
    TknU8   asgnrAsgne;     /* assignor/assignee                       */
    TknU8   logLnkNegot;    /* logical link identifier negotiation     */
    TknU8   mode;           /* mode of operation                       */
    TknU8   multiFrm;       /* multiple frame establishment support    */
    TknU8   hdrNohdr;       /* rate adaption header/no hder            */
    TknU8   parity;         /* parity information                      */
    TknU8   nmbDatBits;     /* number of data bits excluding parity bit */
    TknU8   nmbStpBits;     /* number of stop bits                     */
    TknU8   modemType;      /* modem type                              */
    TknU8   duplexMode;     /* duplex mode                             */
    TknU8   usrInfLyr2Prot; /* user info layer 2 protocol              */
    TknU8   lyr2Ident;      /* layer 2 identity                        */
    TknU8   usrInfLyr3Prot; /* user info layer 3 protocol              */
    TknU8   lyr3Ident;      /* layer 3 identity                        */
} SiUsrServInfo;
```

The fields in the SiUsrServInfo structure are encoded as follows:

Field	Value
infoTranCap	Information transfer capability. Defined values: 0x00 = ITC_SPEECH Speech 0x08 = ITC_UNRDIG Unrestricted digital information 0x09 = ITC_RESDIG Restricted digital information 0x10 = ITC_A31KHZ 3.1 kHz audio 0x11 = ITC_A7KHZ 7 kHz audio 0x12 = ITC_A15KHZ 15 kHz audio (not supported in ANSI or BICC variants) 0x18 = ITC_VIDEO Video
cdeStand	Coding standard. Defined values: 0x00 = CSTD_CCITT CCITT standards 0x01 = CSTD_INT Other international standards 0x02 = CSTD_NAT National standard 0x03 = CSTD_NET Network standard

Field	Value
infoTranRate0	Information transfer rate on the bearer channel from the origination to the destination. Defined values: 0x00 = ITR_PKT Packet mode 0x10 = ITR_64KBIT Circuit mode, 64 kbit/s 0x11 = ITR_2X64KBIT Circuit mode, 2x64 kbit/s 0x13 = ITR_384KBIT Circuit mode, 384 kbit/s 0x14 = ITR_1472KBIT Circuit mode, 1472 kbit/s (ANSI variants only) 0x15 = ITR_1536KBIT Circuit mode, 1536 kbit/s 0x17 = ITR_1920KBIT Circuit mode, 1920 kbit/s 0x18 = ITR_MULIRATE Circuit mode, multi rate (currently not supported)
tranMode	Transfer mode. Defined values: 0x00 = TM_CIRCUIT Circuit mode 0x02 = TM_PACKET Packet mode
establish	Establishment. Defined values for non-BICC variants: 0x00 = E_DEMAND Demand (not supported in BICC) 0x01 = E_PERM Permanent (not supported in BICC)
config	Configuration. Defined values: 0x00 = POINT2POINT Point to point 0x01 = MULTIPOINT Multi-point
chanStruct	Structure. Defined values: 0x00 = S_DEF Default 0x01 = S_8KHZINTEG 8 kHz integrity 0x04 = S_SDUINTEG Service data unit integrity
infoTranRate1	Information transfer rate from the destination to the origination. Defined values: 0x00 = ITR_PKT Packet mode 0x10 = ITR_64KBIT Circuit mode, 64 kbit/s 0x11 = ITR_2X64KBIT Circuit mode, 2x64 kbit/s 0x13 = ITR_384KBIT Circuit mode, 384 kbit/s 0x14 = ITR_1472KBIT Circuit mode, 1472 kbit/s (not supported in BICC) 0x15 = ITR_1536KBIT Circuit mode, 1536 kbit/s 0x17 = ITR_1920KBIT Circuit mode, 1920 kbit/s 0x18 = ITR_MULIRATE Circuit mode, multi rate ((not supported in BICC)
	Symmetry. Defined value: 0x00 = S_BISYM Bi-directional symmetry
usrInfLyr1Prot	User information layer 1 protocol. Defined values: 0x01 = UIL1_CCITTV110 CCITT standardized rate adaptation V.110/X.30 0x02 = UIL1_G711ULAW Recommendation G.711 u-Law 0x03 = UIL1_G711ALAW Recommendation G.711 A-Law 0x04 = UIL1_G721ADCPM Recommendation G.721 32 kbit/s ADCPM and Recommendation I.460 0x05 = UIL1_G722G725 Recommendation G.722 and G.725 - 7 kHz Audio 0x06 = UIL1_H261 Recommendation H.261, 384 kbit/s video 0x07 = UIL1_NONCCITT Non-CCITT standardized rate adaptation 0x08 = UIL1_CCITTV120 CCITT standardized rate adaptation V.120 0x09 = UIL1_CCITTX31 CCITT standardized rate adaptation X.31 HDLC
	Layer 1 identity. Defined value: 0x01 = L1_IDENT Layer 1 identity

Field	Value
usrRate	User rate. Defined values: 0x00 = UR_EINI460 Determined by E bits in I.460 0x01 = UR_600 0.6 kbit/s, V.6 and X.1 0x02 = UR_1200 1.2 kbit/s, V.6 0x03 = UR_2400 2.4 kbit/s, V.6 and X.1 0x04 = UR_3600 3.6 kbit/s, V.6 0x05 = UR_4800 4.8 kbit/s, V.6 and X.1 0x06 = UR_7200 7.2 kbit/s, V.6 0x07 = UR_8000 8.0 kbit/s, I.460 0x08 = UR_9600 9.6 kbit/s, V.6 and X.1 0x09 = UR_14400 14.4 kbit/s, V.6 0x0a = UR_16000 16 kbit/s, I.460 0x0c = UR_32000 32 kbit/s, I.460 0x0e = UR_48000 48 kbit/s, V.6 and X.1 0x0f = UR_56000 56 kbit/s, V.6 0x10 = UR_64000 56 kbit/s, V.6 0x15 = UR_134 .1345 kbit/s, X.1 0x17 = UR_75_1200 .075/1200 kbit/s, V.6 and X.1 0x18 = UR_1200_75 1200/.075 kbit/s, V.6 and X.1 0x19 = UR_50 .050 kbit/s, V.6 and X.1 0x1a = UR_75 .075 kbit/s, V.6 and X.1 0x1b = UR_110 .110 kbit/s, V.6 and X.1 0x1d = UR_200 .200 kbit/s, V.6 and X.1 0x1e = UR_300 .300 kbit/s, V.6 and X.1 0x1f = UR_12000 12 kbit/s, V.6
negot	0x00 = N_IBNOTPOSS Inband not possible 0x01 = N_IBPOSS Inband possible
syncAsync	Synchronous or asynchronous. Defined values: 0x00 = SA_SYNC Synchronous
flcOnRx	Flow control on reception. Defined values: 0x00 = FLCRX_NOTACC Cannot accept data with flow control 0x01 = FLCRX_ACC Can accept data with flow control
flcOnTx	Flow control on transmission. Defined values: 0x00 = FLCTX_NOTREQ Send with flow control not required 0x01 = FLCTX_REQ Required to send with flow control
niClkOnRx	Network independent clock on reception. Defined values 0x00 = NICRX_NOTACC Cannot accept data with nic 0x01 = NICRX_ACC Can accept data with nic
niClkOnTx	Network independent clock on transmission. Defined values 0x01 = NICTX_REQ Required to send with nic

Field	Value
interRate	Intermediate rate. Defined values: 0x00 = IR_NONE None specified 0x02 = IR_16KBIT 16 kbit/s 0x03 = IR_32KBIT 32 kbit/s
inOutBandNeg	Inband or outband negotiation. Defined values: 0x00 = N_OBNOTPOSS Outband not possible 0x01 = N_OBPOSS Outband possible
asgnrAsgne	Assignor or assignee. Defined values: 0x00 = AA_ORGASGNEE Originator is assignee 0x01 = AA_ORGASGNOR Originator is assignor
logLnkNegot	Logical link identifier negotiation. Defined values: 0x00 = LLI_DEF Default 0x01 = LLI_FULLNEG Full protocol negotiation
mode	Mode of operation. Defined values: 0x00 = MOO_BITTRANS Bit transparent 0x01 = MOO_PROTSEN Protocol sensitive
multiFrm	 0x00 = MFE_NOTSUP Not supported 0x01 = MFE_SUP Supported
hdrNohdr	Rate adaption header or no header. Defined values: 0x00 = IRAH_INC Header included 0x01 = IRAH_NOTINC Header not included
parity	 0x00 = PARI_ODD Odd 0x02 = PARI_EVEN Even 0x03 = PARI_NONE None 0x04 = PARI_0 Force to 0 0x05 = PARI_1 Force to 1
nmbDatBit	Number of data bits excluding the parity bit. Defined values: 0x00 = NDB_UNUSED None specified 0x01 = NDB_5 5 bits 0x02 = NDB_7 7 bits
nmbStpBits	 0x00 = NSB_UNUSED None specified 0x01 = NSB_1 1 stop bit 0x02 = NSB_15 1.5 stop bits 0x03 = NSB_2 2 stop bits

Field	Value
modemType	Modem type. Defined values: 0x01 = MODEM_V21 V.21 0x03 = MODEM_V22BIS 0x04 = MODEM_V23 V.23 0x06 = MODEM_V26BIS V.26bis 0x07 = MODEM_V26TER V.26ter 0x09 = MODEM_V27BIS V.27bis 0x0a = MODEM_V27TER V.27ter 0x0b = MODEM_V29 V.29 0x0c = MODEM_V32 V.32
duplexMode	Duplex mode. Defined values: 0x00 = DUPMODE_HALF Half duplex 0x01 = DUPMODE_FULL Full duplex
	User information layer 2 protocol. Defined values for all supported variants except BICC: 0x01 = UIL2_BASIC Basic mode - ISO 1745 0x02 = UIL2_Q921 CCITT Recommendation Q.921 0x06 = UIL2_X25SLP CCITT X.25, single link 0x07 = UIL2_X25MLP CCITT X.25, multi link 0x08 = UIL2_T71 Extended LAPB HDX (CCITT T.71) 0x09 = UIL2_HDLCARM HDLC ARM - ISO 4335 0x0a = UIL2_HDLCNRM HDLC NRM - ISO 4335 0x0b = UIL2_HDLCABM HDLC ABM - ISO 4335 0x0c = UIL2_LANLLC LAN LLC - ISO 8802/2 0x0e = UIL2_Q922 CCITT Recommendation Q.922 0x10 = UIL2_USRSPEC CCITT user specified 0x11 = UIL2_T90 CCITT T.90 Defined values for BICC: 0x02 = UIL2_Q921 CCITT Recommendation Q.921 0x06 = UIL2_X25SLP CCITT X.25, single link
lyr2Ident	Layer 2 identity. Defined value: 0x02 = L2_IDENT Layer 2 identity
usrInfLyr3Prot	User information layer 3 protocol. Defined values for all supported variants except BICC: 0x05 = UIL3_T90 CCITT T.90 0x06 = UIL3_X25PLP CCITT X.25, packet layer 0x07 = UIL3_ISO8208 ISO 8208 0x08 = UIL3_ISO8348 ISO 8348 0x09 = UIL3_ISO8473 ISO 8473 0x0a = UIL3_T70 CCITT Recommendation T.70 0x10 = UIL3_USRSPEC CCITT user specified Defined values for BICC: 0x02 = UIL3_Q931 CCITT Recommendation Q.931 0x06 = UIL3_X25PLP CCITT X.25, packet layer
lyr3Ident	Layer 3 identity. Defined value: 0x00 = L3_IDENT Layer 3 identity

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88		ANSI 95	BICC	ETSI V2	ETSI V3
infoTranCap	*	*	*	*	*	*
cdeStand	*	*	*	*	*	*
infoTranRate0	*	*	*	*	*	*
tranMode	*	*	*	*	*	*
establish	*	*	*	*	*	*
config	*	*	*	*	*	*
chanStruct	*	*	*	*	*	*
infoTranRate1	*	*	*	*	*	*
symmetry	*	*	*	*	*	*
usrInfLyrlProt	*	*	*	*	*	*
lyr1Ident	*	*	*	*	*	*
usrRate	*	*	*	*	*	*
negot		*	*	*	*	*
syncAsync		*	*	*	*	*
flcOnRx		*	*	*	*	*
flcOnTx		*	*	*	*	*
niClkOnRx		*	*	*	*	*
niClkOnTx		*	*	*	*	*
interRate		*	*	*	*	*
inOutBandNeg		*	*		*	*
asgnrAsgne		*	*		*	*
logLnkNegot		*	*		*	*
mode		*	*		*	*
multiFrm		*	*		*	*
hdrNohdr		*	*		*	*
parity		*	*	*	*	*
nmbDatBits		*	*	*	*	*
nmbStpBits		*	*	*	*	*
modemType		*	*	*	*	*
duplexMode		*	*	*	*	*
usrInfLyr2Prot	*	*	*	*	*	*
lyr2Ident	*	*	*	*	*	*
usrInfLyr3Prot	*	*	*	*	*	*
lyr3Ident	*	*	*	*	*	*

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White	ITU 97	Q.767
infoTranCap	*	*	*	*
cdeStand	*	*	*	*
infoTranRate0	*	*	*	*
tranMode	*	*	*	*
establish	*	*	*	*
config	*	*	*	*
chanStruct	*	*	*	*
infoTranRate1	*	*	*	*
symmetry	*	*	*	*
usrInfLyrlProt	*	*	*	*
lyr1Ident	*	*	*	*
usrRate	*	*	*	*
negot	*	*	*	*
syncAsync	*	*	*	*
flcOnRx	*	*	*	*
flcOnTx	*	*	*	*
niClkOnRx	*	*	*	*
niClkOnTx	*	*	*	*
interRate	*	*	*	*
inOutBandNeg	*	*	*	*
asgnrAsgne	*	*	*	*
logLnkNegot	*	*	*	*
mode	*	*	*	*
multiFrm	*	*	*	*
hdrNohdr	*	*	*	*
parity	*	*	*	*
nmbDatBits	*	*	*	*
nmbStpBits	*	*	*	*
modemType	*	*	*	*
duplexMode	*	*	*	*
usrInfLyr2Prot	*	*	*	*
lyr2Ident	*	*	*	*
usrInfLyr3Prot	*	*	*	*
lyr3Ident	*	*	*	*

User-to-user information

Associated variants: ANSI 88, ANSI 95, BICC, ETSI V2, ETSI V3, ITU Blue, ITU White, ITU 97

Passes user-to-user data transparently to the end user at the far end of the connection.

```
typedef struct _usr2UsrInfo  /* User to user information */
{
    ElmtHdr eh;              /* element header           */
    TknStr  info;            /* user to user information */
} SiUsr2UsrInfo;
```

The info field is binary data passed transparently to the far-end user.

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 88	ANSI 95	BICC	ETSI V2	ETSI V3
info	*	*	*	*	*

Tokens for the ITU variants

Token	ITU Blue	ITU White	ITU 97
info	*	*	*

User-to-user indicators

Associated variants: ANSI 95, BICC, ETSI V2, ETSI V3, ITU Blue, ITU White, ITU 97, Q.767

Contains information sent in a request, or a response to a request for user-to-user signaling supplementary services.

```
typedef struct _usr2UsrInd  /* User to User indicators   */
{
    ElmtHdr eh;             /* element header           */
    TknU8   type;           /* type                     */
    TknU8   serv1;          /* service 1                */
    TknU8   serv2;          /* service 2                */
    TknU8   serv3;          /* service 3                */
    TknU8   spare;          /* spare bits               */
    TknU8   netDscrdInd;    /* network discard indicator */
} SiUsr2UsrInd;
```

The fields in the SiUsr2UsrInd structure are encoded as follows:

<table>
<tr><th>Field</th><th>Value</th></tr>
<tr><td>type</td><td>Type of indicator. Defined values:
0 = Request
1 = Response</td></tr>
<tr><td>serv1

serv3</td><td>Defined values:
<table>
<tr><th></th><th>Meaning in request</th><th>Meaning in response</th></tr>
<tr><td>0</td><td>No information</td><td>No information</td></tr>
<tr><td>1</td><td></td><td>Not provided</td></tr>
<tr><td>2</td><td>Request, not essential</td><td>Provided</td></tr>
<tr><td>3</td><td>Request, essential</td><td>Spare</td></tr>
</table></td></tr>
<tr><td>netDscrdInd</td><td>Network discard indicator. Defined values:
0 = No information
1 = User-to-user information discarded by the network</td></tr>
</table>

Tokens for the ANSI, BICC, and ETSI variants

Token	ANSI 95	BICC	ETSI V2	ETSI V3
type	*	*	*	*
serv1	*	*	*	*
serv2	*	*	*	*
serv3	*	*	*	*
spare				
netDscrdInd	*	*	*	*

Tokens for the ITU and Q.767 variants

Token	ITU Blue	ITU White	ITU 97	Q.767
type	*	*	*	*
serv1	*	*	*	*
serv2	*	*	*	*
serv3	*	*	*	*
spare	H			
netDscrdInd		*	*	*

Information elements for Japan/NTT variant

The following information elements are specific to the NTT variant of the ISUP layer:

- Additional user identification
- Calling number non-notification reason
- Carrier information transfer
- Charge information
- Charge information delay
- Charge information type
- Contractor number
- Message area information

Additional user identification

```
typedef struct _addUsrId
{
    ElmtHdr eh;                      /* element header */
    TknStr  usrId;
} SiAddUsrId;
```

Calling number non-notification reason

```
typedef struct _cgNumNonNotRsn
{
    ElmtHdr eh;                    /* element header          */
    TknU8   nonNotRsn;             /* non notification reason */
} SiCgNumNonNotRsn;
```

Carrier information transfer

```
typedef struct _carrierInfoTrans  /* Carrier Information Transfer */
{
    ElmtHdr eh;
    TknU8   infoType;
    TknU8   spare;
    TknStr  carrierInfo;
} SiCarrierInfoTrans;
```

Charge information

```
typedef struct _chargeInfo       /* Charge Information */
{
    ElmtHdr eh;
    TknStr  chargeInfo;
} SiChargeInfo;
```

Charge information delay

```
typedef struct _chargeInfoDly    /* Charge Information Delay */
{
    ElmtHdr eh;
    TknStr  chargeInfo;
} SiChargeInfoDly;
```

Charge information type

```
typedef struct _chargeInfoType   /* Charge Information Type */
{
    ElmtHdr eh;
    TknU8   chargeInfoType;
} SiChargeInfoType;
```

Contractor number

```
typedef struct _contractorNum    /* Contractor Number             */
{
    ElmtHdr eh;                  /* element header                */
    TknU8   natAddr;             /* nature of address indicator   */
    TknU8   oddEven;             /* odd or even address signal    */
    TknU8   spare1;              /* spare bit 7, octet 2          */
    TknU8   numPlan;             /* numbering plan                */
    TknU8   spare2;              /* spare bits 1-4, octet 2       */
    TknStr  addrSig;             /* addressing signal             */
} SiContractorNum;
```

Message area information

```
typedef struct _msgAreaInfo      /* Message Area Information */
{
    ElmtHdr eh;
    TknU8   oddEven;
    TknU8   infoInd;
    TknStr  info;
} SiMsgAreaInfo;
```

11 Message and function cross-reference

Sending ISUP protocol messages

The following table describes the event initialization routines and function combinations used to send various ISUP protocol messages. When applicable, the value required for the ***evntType*** argument to the function is shown.

Message type	Mnemonic	routine	Function
Address complete	ACM	**ISUPInitACM**	**ISUPConnectStatusReq** evntType = ADDRCMPLT
Answer	ANM	**ISUPInitANM**	**ISUPConnectResp**
Application transport message	APM	none	**ISUPStatusReq** evntType = APPTRANSPORT
Blocking (not supported by BICC)	BLO	none	**ISUPStatusReq** evntType = CIRBLKREQ
Blocking acknowledgment (not supported by BICC)	BLA	none	**ISUPStatusReq** evntType = CIRBLKRSP
Call modification complete	CMC	none	**ISUPConnectStatusReq** evntType = MODCMPLT
Call modification reject	CMRJ	none	**ISUPConnectStatusReq** evntType = MODREJ
Call modification request	CMR	none	**ISUPConnectStatusReq** evntType = MODIFY
Call progress	CPG	**ISUPInitCPG**	**ISUPConnectStatusReq** evntType = PROGRESS
Charge information	CRG	none	Unsupported.
Circuit group blocking	CGB	none	**ISUPStatusReq** evntType = CIRGRPBLKREQ
Circuit group blocking acknowledgment	CGBA	none	**ISUPStatusReq** evntType = CIRGRPBLKRSP
Circuit group query	CQM	none	**ISUPStatusReq** evntType = CIRGRPQRYREQ
Circuit group query response	CQR	none	Generated automatically by ISUP task.
Circuit group reset	GRS	none	**ISUPStatusReq** evntType = CIRGRPRESREQ

Circuit group reset acknowledgment	GRA	none	Generated automatically by ISUP task.
Circuit group unblocking	CGU	none	**ISUPStatusReq** evntType = CIRGRPUNBLKREQ
Circuit group unblocking acknowledgment	CGUA	none	**ISUPStatusReq** evntType = CIRGRPUNBLKRSP
Circuit reservation	CRM	**ISUPInitCRM**	**ISUPStatusReq** evntType = CIRRESERVE
Circuit reservation acknowledgment	CRA	none	**ISUPStatusReq** evntType = CIRRESERVEACK
Circuit validation response	CVR	none	Generated automatically by ISUP task.
Circuit validation test	CVT	none	**isupValidateCircuit** (ISUP management functions)
Confusion	CFN	none	Generated automatically by ISUP task.
Connect	CON	**ISUPInitCON**	**ISUPConnectResp**
Continuity	COT	**ISUPInitCOT**	**ISUPStatusReq** evntType = CONTREP
Continuity check request (not supported by BICC	CCR	none	**ISUPStatusReq** evntType = CONTCHK
Delayed release	DRS	none	Unsupported.
Exit	EXM	none	Generated automatically by ISUP task.
Facility	FAC	**ISUPInitFAC**	**ISUPFacilityReq** evntType = FACILITY
Facility accepted	FAA	**ISUPInitFAA**	**ISUPFacilityReq** evntType = FACILITYACC
Facility reject	FRJ	**ISUPInitFRJ**	**ISUPFacilityReq** evntType = FACILITYREJ
Facility request	FAR	**ISUPInitFAR**	**ISUPFacilityReq** evntType = FACILITYREQ
Facility deactivate	FAR	**ISUPInitFAD**	**ISUPFacilityReq** evntType = FACILITYDEACT
Facility information	FAR	**ISUPInitFAI**	**ISUPFacilityReq** evntType = FACILITYINFO
Forward transfer	FOT	**ISUPInitFOT**	**ISUPConnectStatusReq** evntType = FRWDTRSFR
Identification request	IDM	none	**ISUPConnectStatusReq** evntType = IDENTRSP

Identification response	IDR	none	**ISUPConnectStatusReq** evntType = IDENTREQ
Information	INF	**ISUPInitINF**	**ISUPConnectStatusReq** evntType = INFORMATION
Information request	INR	**ISUPInitINR**	**ISUPConnectStatusReq** evntType = INFORMATREQ
Initial address	IAM	**ISUPInitIAM**	**ISUPConnectReq**
Loop prevention	LOP	none	**ISUPStatusReq** evntType = LOOPPREVENTION
Loopback acknowledgment (not supported by BICC)	LPA	none	**ISUPStatusReq** evntType = LOOPBCKACK
Network resource manager	NRM	none	**ISUPConnectStatusReq** evntType = NETRESMGR
Overload (not supported by BICC)	OLM	none	Generated automatically by ISUP task.
Pass-along (not supported by BICC)	PAM		**ISUPDataReq**
Pre-release information	PRI	none	**ISUPStatusReq** evntType = PRERELEASE
Release	REL	**ISUPInitREL**	**ISUPReleaseReq**
Release complete	RLC	none	**ISUPReleaseResp**
Reset circuit	RSC	none	**ISUPStatusReq** evntType = CIRRESREQ
Resume	RES	**ISUPInitRES**	**ISUPResumeReq**
Subsequent address	SAM	**ISUPInitSAM**	**ISUPConnectStatusReq** evntType = SUBSADDR
Suspend	SUS	**ISUPInitSUS**	**ISUPSuspendReq**
Unblocking (not supported by BICC)	UBL	none	**ISUPStatusReq** evntType = CIRUNBLKREQ
Unblocking acknowledgment (not supported by BICC)	UBA	none	**ISUPStatusReq** evntType = CIRUNBLKRSP
Unequipped circuit identification code	UCIC	none	Generated automatically by ISUP task.
User part available (not supported by BICC)	UPA	none	Generated automatically by ISUP task.
User part test (not supported by BICC)	UPT	none	Generated automatically by ISUP task.
User-to-user information	USR	none	**ISUPDataReq**

Receiving ISUP protocol messages

The following table shows the ***evntType/indType*** combinations returned when receiving ISUP protocol messages:

Message type	Mnemonic	indType	evntType
Address complete	ACM	EVTSITCNSTIND	ADDRCMPLT
Answer	ANM	EVTSITCONCFM	None
Application transport message	APM	EVTSITSTAIND	APPTRANSPORT
Blocking (not supported in BICC)	BLO	EVTSITSTAIND	CIRBLKREQ
Blocking acknowledgment (not supported in BICC)	BLA	EVTSITSTAIND	CIRBLKRSP
Call modification complete	CMC	EVTSITCNSTIND	MODCMPLT
Call modification reject	CMRJ	EVTSITCNSTIND	MODREJ
Call modification request	CMR	EVTSITCNSTIND	MODIFY
Call progress	CPG	EVTSITCNSTIND	PROGRESS
Charge information	CRG	N/A	Unsupported.
Circuit group blocking	CGB	EVTSITSTAIND	CIRGRPBLKREQ
Circuit group blocking acknowledgment	CGBA	EVTSITSTAIND	CIRGRPBLKRSP
Circuit group query	CQM	N/A	Handled automatically by ISUP task.
Circuit group query response	CQR	EVTSITSTAIND	CIRGRPQRYRSP
Circuit group reset	GRS	EVTSITSTAIND	CIRGRPRESREQ
Circuit group reset acknowledgment	GRA	EVTSITSTAIND	CIRGRPRESACK
Circuit group unblocking	CGU	EVTSITSTAIND	CIRGRPUNBLKREQ
Circuit group unblocking acknowledgment	CGUA	EVTSITSTAIND	CIRGRPUNBLKRSP
Circuit reservation	CRM	EVTSITSTAIND	CIRRESERVE
Circuit reservation acknowledgment	CRA	EVTSITSTAIND	CIRRESERVEACK
Circuit validation response	CVR	N/A	Handled automatically by ISUP task.
Circuit validation test	CVT	N/A	Handled automatically by ISUP task.
Confusion	CFN	EVTSITSTAIND	CONFUSION
Connect	CON	EVTSITCONCFM	
Continuity	COT	EVTSITSTAIND	CONTREP
Continuity check request (not supported in BICC)	CCR	EVTSITSTAIND	CONTCHK
Delayed release	DRS	N/A	Unsupported.
Exit	EXM	EVTSITCNSTIND	EXIT
Facility	FAC	EVTSITFACIND	FACILITY

Facility accepted	FAA	EVTSITFACIND	FACILITYACC
Facility reject	FRJ	EVTSITFACIND	FACILITYREJ
Facility request	FAR	EVTSITFACIND	FACILITYEQ
Facility deactivate	FAD	EVTSITFACIND	FACILITYDEACT
Facility information	FAI	EVTSITFACIND	FACILITYINFO
Forward transfer	FOT	EVTSITCNSTIND	FRWDTRSFR
Identification request	IDM	EVTSITCNSTIND	IDENTRSP
Identification response	IDR	EVTSITCNSTIND	IDENTREQ
Information	INF	EVTSITCNSTIND	INFORMATION
Information request	INR	EVTSITCNSTIND	INFORMATREQ
Initial address	IAM	EVTSITCONIND	
Loop prevention	LOP	EVTSITSTAIND	LOOPPREVENTION
Loopback acknowledgment (not supported by BICC)	LPA	EVTSITSTAIND	LOOPBCKACK
Network resource manager	NRM	EVTSITCNSTIND	NETRESMGR
Overload (not supported in BICC)	OLM	N/A	Handled automatically by ISUP task.
Pass-along (not supported in BICC)	PAM	EVTSITDATIND	
Release	REL	EVTSITRELIND	
Release complete	RLC	EVTSITRELCFM	
Reset circuit	RSC	EVTSITSTAIND	CIRRESREQ
Resume	RES	EVTSITRESMIND	
Subsequent address	SAM	EVTSITCNSTIND	SUBSADDR
Suspend	SUS	EVTSITSUSPIND	
Unblocking (not supported in BICC)	UBL	EVTSITSTAIND	CIRUNBLKREQ
Unblocking acknowledgment (not supported in BICC)	UBA	EVTSITSTAIND	CIRUNBLKRSP
Unequipped circuit identification code	UCIC	EVTSITSTAIND	CIRUNEQPD
User part available (not supported in BICC)	UPA	N/A	Handled automatically by ISUP task.
User part test (not supported in BICC)	UPT	N/A	Handled automatically by ISUP task.
User-to-user information	USR	EVTSITDATIND	

12 Status indications

Status indications overview

This section presents the event types (evntType) received by an application when an EVTSITSTAIND event is received.

Error

For ERROR status indications, the cause token of the cause diagnostic (***causDgn***) information element contains one of the following values:

Cause value	Description
CCNORTTOTSFNET	ISUP layer could not find a route matching the transient network ID supplied in the request. Not generated if the application performs circuit selection.
CCNORTTODEST	the request, or neither the transient network ID nor the called party address were supplied. Not generated if the application performs circuit selection.
CCNOCIRCUIT	ISUP layer could not find a circuit matching the transmission medium and/or ISDN user profiles supplied in the request. Not generated if the application performs circuit selection.
CCINVALCALLREF	Values supplied for a service user instance ID and/or service provider instance ID are invalid.
CCSWTCHCONG	No call references were available to satisfy the request, or the requested circuit is busy.
CCRESCUNAVAIL	There is insufficient memory for the ISUP layer to satisfy the request.
CCREQUNAVAIL	Requested circuit is not configured.
CCDESTOUTORD	Requested circuit was marked unavailable due to receipt of a pause or user part unavailable status indication from MTP.
CCPROTERR	Circuit ID supplied is invalid for the supplied service provider instance ID, the request is invalid for the configured switch type, or the request is invalid for the current circuit or group state.
CCINVALPARAMCONT	The value specified in an information element is invalid.
CCINFOELMSSG	A mandatory information element was not supplied.

Reattempt

The following cause values are associated with reattempt status indications:

Cause value	Meaning
CCREQUNAVAIL	If the application selected the circuit, this value indicates that the requested circuit is not configured. If ISUP is performing circuit selection, this value indicates that no circuit is available.
CCSWTCHCONG	An overload message was received from the network, or the requested circuit is busy or reserved.
CCRESCUNAVAIL	The outbound call lost a glare situation and should be retried on another circuit. The inbound call indication follows this event.

Continuity check

A continuity check (CONTCHK) status indication is delivered to the application when the ISUP layer receives a continuity check request message (CCR) from the far exchange.

When the application receives this indication, it must connect a continuity check loop to the referenced circuit. The application indicates the loop is in place by sending a loopback acknowledgment using **ISUPStatusReq**.

Note: Continuity checks status indications are not supported in BICC.

Continuity report

A continuity report (CONTREP) status indication is delivered to the application when the ISUP layer receives a continuity report message (COT) from the far exchange. When the application receives this indication, it must remove the continuity check loop from the referenced circuit.

Note: Continuity report status indications are not supported in BICC.

Stop continuity

A stop continuity (STPCONTIN) status indication is delivered to the application when the ISUP layer receives a release message (REL) from the far exchange during continuity testing. When the application receives this indication, it must remove the continuity check loop and tone from the referenced circuit.

Note: Stop continuity status indications are not supported in BICC.

Loop back acknowledgment

A loop back acknowledgment (LOOPBCKACK) status indication is delivered to the application when the ISUP layer receives a loop back acknowledgment message (LPA) from the far exchange.

When the application receives this indication, it must check the transmission path of the referenced circuit. The application then indicates the completion of the continuity check by sending a continuity report message using **ISUPStatusReq**.

Note: Loop back acknowledgement status indications are not supported in BICC.

Confusion indication

A confusion (CONFUSION) status indication is delivered to the application when the ISUP layer receives a confusion message (CFN) from the far exchange.

Circuit reservation request

A circuit reservation (CIRRESERVE) indication is delivered to the application when the ISUP layer receives a circuit reservation message (CRM) from the far exchange.

When the application receives this indication, it must consider the referenced circuit to be busy. The application then indicates the receipt of the circuit reservation message by sending a circuit reservation acknowledgment message using **ISUPStatusReq**.

Circuit reservation acknowledgment

A circuit reservation acknowledgment (CIRRESERVEACK) indication is delivered to the application when the ISUP layer receives a circuit reservation acknowledgment message (CRA) from the far exchange. The application must initiate a connection on the referenced circuit before the far exchange times out.

Circuit group query response

A circuit group query response (CIRGRPQRYRSP) indication is delivered to the application when the ISUP layer receives a circuit group query response message (CQR) from the far exchange. The rangStat and cirStateInd members of the status event (SiStaEvnt) structure are populated when CIRGRPQRYRSP indications are delivered to the application.

Circuit block request

A circuit block request (CIRBLKREQ) indication is delivered to the application when the ISUP layer receives a blocking message (BLO) from the far exchange. The application must consider the referenced circuit to be remotely blocked. The application must then acknowledge the blocking message by sending a blocking acknowledgment message using **ISUPStatusReq**.

Note: Circuit block request status indications are not supported in BICC.

Circuit block response

A circuit block response (CIRBLKRSP) indication is delivered to the application when the ISUP layer receives a blocking acknowledgment message (BLA) from the far exchange. The application must consider the referenced circuit to be locally blocked.

Note: Circuit block response status indications are not supported in BICC.

Circuit unblock request

A circuit unblock request (CIRUNBLKREQ) indication is delivered to the application when the ISUP layer receives an unblocking message (UBL) from the far exchange. The application must consider the referenced circuit to be remotely unblocked. The application must then acknowledge the unblocking message by sending an unblocking acknowledgment message using **ISUPStatusReq**.

Note: Circuit unblock request status indications are not supported in BICC.

Circuit unblock response

A circuit unblock response (CIRUNBLKRSP) indication is delivered to the application when the ISUP layer receives an unblocking acknowledgment message (UBA) from the far exchange. The application must consider the referenced circuit to be locally unblocked.

Note: Circuit unblock response status indications are not supported in BICC.

Circuit reset request

A circuit reset (CIRRESREQ) indication is delivered to the application when the ISUP layer receives a reset message (UBL) from the far exchange, or a reset was initiated by ISUP management. The application must consider the referenced circuit to be idle.

Circuit group block request

A circuit group block request (CIRGRPBLKREQ) indication is delivered to the application when the ISUP layer receives a circuit group blocking message (CGB) from the far exchange. The application must consider the referenced circuits to be remotely blocked. The application must then acknowledge the circuit group blocking message by sending a circuit group blocking acknowledgment message using **ISUPStatusReq**.

The rangStat member of the status event (SiStaEvnt) structure is populated when CIRGRPBLKREQ indications are delivered to the application.

Circuit group block response

A circuit group block response (CIRGRPBLKRSP) indication is delivered to the application when the ISUP layer receives a circuit group blocking acknowledgment message (CGBA) from the far exchange. The application must consider the referenced circuits to be locally blocked.

The rangStat member of the status event (SiStaEvnt) structure is populated when CIRGRPBLKRSP indications are delivered to the application.

Circuit group unblock request

A circuit group unblock request (CIRGRPUNBLKREQ) indication is delivered to the application when the ISUP layer receives a circuit group unblocking message (CGU) from the far exchange. The application must consider the referenced circuits to be remotely unblocked. The application must then acknowledge the circuit group unblocking message by sending a circuit group unblocking acknowledgment message using **ISUPStatusReq**.

The rangStat member of the status event (SiStaEvnt) structure is populated when CIRGRPUNBLKREQ indications are delivered to the application.

Circuit group unblock response

A circuit group unblock response (CIRGRPUNBLKRSP) indication is delivered to the application when the ISUP layer receives a circuit group unblocking acknowledgment message (CGUA) from the far exchange. The application must consider the referenced circuits to be locally unblocked.

The rangStat member of the status event (SiStaEvnt) structure is populated when CIRGRPUNBLKRSP indications are delivered to the application.

Circuit unequipped

A circuit unequipped (CIRUNEQPD) status indication is delivered to the application when the ISUP layer receives a unequipped CIC message (UCIC) from the far exchange.

MTP pause indication

One or more MTP pause (MTPPAUSE) indications are delivered to the application when the ISUP layer receives a pause indication from the MTP 3 layer. The rangStat member of the status event structure (SiStaEvnt) is populated when MTPPAUSE indications are delivered to the application.

The range value indicates the number of affected circuits minus one. The circuit member of the IsupRcvInfoBlk indicates the first affected circuit. Affected circuits are considered to be out of service.

MTP backup indication

One or more MTP backup (MTPBACKUP) indications are delivered to the application when the ISUP layer receives a backup indication from the MTP 3 layer. This can occur when the application binds to an ISUP user SAP.

The MTP backup status indicates that the run state of MTP is currently backup on this board in a redundant board pair, monitoring the status of the primary. No active traffic passes through this SAP until the board becomes the primary member of the pair.

MTP primary indication

One or more MTP primary (MTPPRIMARY) indications are delivered to the application when the ISUP layer receives a primary indication from the MTP 3 layer. This can occur when the application binds to an ISUP user SAP.

The MTP backup status indicates that the run state of MTP is currently backup on this board in a redundant board pair, monitoring the status of the primary. No active traffic passes through this SAP until the board becomes the primary member of the pair.

MTP resume indication

One or more MTP resume (MTPRESUME) indications are delivered to the application when the ISUP layer receives a resume indication from the MTP3 layer. The rangStat member of the status event structure (SiStaEvnt) is populated when MTPRESUME indications are delivered to the application.

The range value indicates the number of affected circuits minus one. The circuit member of the IsupRcvInfoBlk indicates the first affected circuit. Affected circuits are considered to be idle.

MTP standalone indication

One or more MTP standalone (MTPSTANDALONE) indications are delivered to the application when the ISUP layer receives a standalone indication from the MTP 3 layer. This can occur when the application binds to an ISUP user SAP.

The MTP standalone status indicates that the application is in a non-redundant configuration, and that normal operation can begin.

Remote user unavailable indication

One or more remote user unavailable (RMTUSRUNAVAIL) indications are delivered to the application when the ISUP layer receives a remote user unavailable indication from the MTP 3 layer. The rangStat member of the status event structure (SiStaEvnt) is populated when RMTUSRUNAVAIL indications are delivered to the application.

The range value indicates the number of affected circuits minus one. The circuit member of the IsupRcvInfoBlk indicates the first affected circuit. Affected circuits are considered to be out of service.

Remote user available indication

One or more remote user available (RMTUSRAVAIL) indications are delivered to the application when the ISUP layer detects that the remote user is available. The rangStat member of the status event structure (SiStaEvnt) is populated when RMTUSRAVAIL indications are delivered to the application.

The range value indicates the number of affected circuits minus one. The circuit member of the IsupRcvInfoBlk indicates the first affected circuit. Affected circuits are considered to be idle.

MTP congestion indication

One or more MTP congestion (MTPCONGEST) indications are delivered to the application when the ISUP layer receives a congestion indication from the MTP 3 layer. The rangStat member of the status event structure (SiStaEvnt) is populated when MTPCONGEST indications are delivered to the application.

The range value indicates the number of affected circuits minus one. The circuit member of the IsupRcvInfoBlk indicates the first affected circuit. The application must avoid using affected circuits until an MTP stop congestion (MTPSTOPCONGEST) is received.

MTP stop congestion indication

One or more MTP stop congestion (MTPSTOPCONGEST) indications are delivered to the application when the ISUP layer receives a stop congestion indication from the MTP 3 layer. The rangStat member of the status event structure (SiStaEvnt) is populated when MTPSTOPCONGEST indications are delivered to the application.

The range value indicates the number of affected circuits minus one. The circuit member of the IsupRcvInfoBlk indicates the first affected circuit.

Index