Dialogic®

Making Innovation Thrive™

Dialogic® NaturalAccess™ ISDN Software Supplementary Services Developer's Manual

December 2009 64-0510-02

www.dialogic.com

Copyright and legal notices

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

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

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

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

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

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

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

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

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

Revision history

Revision	Release date	Notes
9000-6502-10	August, 1998	Colin Ferguson, Rick Ringel
9000-6502-11	October, 1998	CYF Final Release (Beta Release 3)
9000-6502-12	November, 1998	CYF Final Release
9000-6502-13	February, 1999	CYF changes for Natural Access
9000-6502-14	September, 1999	CYF changes for Natural Access 3.0
9000-6502-15	December, 1999	MCM conversion to Frame
9000-6502-16	July, 2000	EPS/SJC, Natural Access 3.0 and 4.0
9000-6502-17	March, 2001	SJC, for NACD 2000-2
9000-6502-18	April, 2001	SJC, for NACD 2001-1 Beta
9000-6502-19	August, 2001	SJC, for NACD 2001-1
9000-6502-20	November, 2001	SJC, for NACD 2002-1 Beta
9000-6502-21	May, 2002	LBG, NACD 2002-1
9000-6502-22	November, 2002	LBG, Natural Access 2003-1 Beta
9000-6502-23	April, 2003	LBG, Natural Access 2003-1
9000-6502-24	April, 2004	SRR, Natural Access 2004-1
9000-6502-25	October, 2005	DEH, Natural Access 2005-1, SP 1
64-0510-01	October, 2009	LBG, NaturalAccess R9.0
64-0510-02	November, 2009	LBG, NaturalAccess R9.0.1
Last modified: December 2, 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™ ISDN Software Supplementary Services Developer's Manual* describes ISDN supplementary services that can be accessed using NaturalAccess ISDN Software. It provides:

- Background information about each available service, as it is defined in the specification for any applicable variants.
- A programming guide for NaturalAccess ISDN Messaging applications.
- Documentation of a demonstration program included with the software.

This document is intended for developers of telephony and voice applications who are using NaturalAccess. This document defines telephony terms where applicable, but assumes that the reader is familiar with basic telephony concepts and the C programming language.

Although some introductory material is provided, this manual also assumes that the reader is familiar with the operation of basic ISDN supplementary services, as described in appropriate specifications.

2 Terminology

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	Dialogic terminology
CG 6060 Board	Dialogic® CG 6060 PCI Media Board
CG 6060C Board	Dialogic® CG 6060C CompactPCI Media Board
CG 6565 Board	Dialogic® CG 6565 PCI Media Board
CG 6565C Board	Dialogic® CG 6565C CompactPCI Media Board
CG 6565e Board	Dialogic® CG 6565E PCI Express Media Board
CX 2000 Board	Dialogic® CX 2000 PCI Station Interface Board
CX 2000C Board	Dialogic® CX 2000C CompactPCI Station Interface Board
AG 2000 Board	Dialogic® AG 2000 PCI Media Board
AG 2000C Board	Dialogic® AG 2000C CompactPCI Media Board
AG 2000-BRI Board	Dialogic® AG 2000-BRI Media Board
NMS OAM Service	Dialogic® NaturalAccess™ OAM API
NMS OAM System	Dialogic® NaturalAccess™ OAM System
NMS SNMP	Dialogic® NaturalAccess™ SNMP API
Natural Access	Dialogic® NaturalAccess™ Software
Natural Access Service	Dialogic® NaturalAccess™ Service
Fusion	Dialogic® NaturalAccess™ Fusion™ VoIP API
ADI Service	Dialogic® NaturalAccess™ Alliance Device Interface API
CDI Service	Dialogic® NaturalAccess™ CX Device Interface API
Digital Trunk Monitor Service	Dialogic® NaturalAccess™ Digital Trunk Monitoring API
MSPP Service	Dialogic® NaturalAccess™ Media Stream Protocol Processing API
Natural Call Control Service	Dialogic® NaturalAccess™ NaturalCallControl™ API
NMS GR303 and V5 Libraries	Dialogic® NaturalAccess™ GR303 and V5 Libraries
Point-to-Point Switching Service	Dialogic® NaturalAccess™ Point-to-Point Switching API
Switching Service	Dialogic® NaturalAccess™ Switching Interface API

Former terminology	Dialogic terminology
Voice Message Service	Dialogic® NaturalAccess™ Voice Control Element API
NMS CAS for Natural Call Control	Dialogic® NaturalAccess™ CAS API
NMS ISDN	Dialogic® NaturalAccess™ ISDN API
NMS ISDN for Natural Call Control	Dialogic® NaturalAccess™ ISDN API
NMS ISDN Messaging API	Dialogic® NaturalAccess™ ISDN Messaging API
NMS ISDN Supplementary Services	Dialogic® NaturalAccess™ ISDN API Supplementary Services
NMS ISDN Management API	Dialogic® NaturalAccess™ ISDN Management API
NaturalConference Service	Dialogic® NaturalAccess™ NaturalConference™ API
NaturalFax	Dialogic® NaturalAccess™ NaturalFax™ API
SAI Service	Dialogic® NaturalAccess™ Universal Speech Access API
NMS SIP for Natural Call Control	Dialogic® NaturalAccess™ SIP API
NMS RJ-45 interface	Dialogic® MD1 RJ-45 interface
NMS RJ-21 interface	Dialogic® MD1 RJ-21 interface
NMS Mini RJ-21 interface	Dialogic® MD1 Mini RJ-21 interface
NMS Mini RJ-21 to NMS RJ-21 cable	Dialogic® MD1 Mini RJ-21 to MD1 RJ-21 cable
NMS RJ-45 to two 75 ohm BNC splitter cable	Dialogic® MD1 RJ-45 to two BNC pairs splitter cable
NMS signal entry panel	Dialogic® Signal Entry Panel

3 NMS ISDN supplementary services

NMS ISDN supplementary services overview

ISDN supplementary services enable an NMS ISDN application to implement powerful functionality beyond basic call control. These functions include:

- Call transfer operations: an application can reroute a call to another user.
- Call forwarding operations: a network or ISDN stack can reroute a call destined for a user to another user.
- Call hold and retrieve services: an application can place a call on hold, and retrieve a held call.
- Advice of charge services: an application can track the costs of a specific call, in real-time.
- Tandem services: an application can support a transit node (one that exists between two other nodes, and route messages from one node to the other).
- Identification services: called or calling parties can exchange information.

Your NMS ISDN application can access supplementary services using the standard NMS ISDN Messaging functions. Supplementary services messages are included in standard ACU messages sent to or received from the NMS ISDN protocol stack. Each service specification and any associated data is stored in an extended data area appended to the ACU message buffer. You can invoke supplementary services with any primitive passed over the ACU interface.

One primitive may contain multiple extended data structures. This allows the application to invoke multiple services with one primitive, or to receive multiple indications with one primitive coming up from the ACU interface.

Some transfer supplementary services can also be accessed using NMS Natural Call Control.

Supplementary service operation summary

The following table lists the supplementary service operations and the variants those operations support. Operations vary depending on the implementation. In this table, V = *isdngen* implementation, X = original variant-specific implementation, and VX = both implementations:

Supplementary service operation	ETSI	Q.SIG	NI2	DMS
Invoke bridge calls	-	X	-	-
Invoke call hold	X	-	-	-
Invoke call retrieve	X	-	-	-
Notify hold	X	X	-	-
Notify retrieve	X	X	-	-
Explicit call transfer	V X	-	-	-
Notify transfer	X	X	-	-
Invoke two channel transfer	-	V	X	X
Notify two channel transfer	-	-	X	-
Release link trunk	-	-	-	X
Invoke call diversion	-	X	-	-
Activate diversion	X	-	-	-
Deactivate diversion	X	-	-	-
Enquire diversion	X	-	-	-
Remind diversion	X	-	-	-
Notify diversion (Q.SIG and ETS 300)	X	X	-	-
Invoke call deflection	X	-	-	-
Activate deflection	X	-	-	-
Deactivate deflection	X	-	-	-
Advice of charge request	X	-	-	-
Advice of charge inform	X	-	-	-
Calling name identification presentation (CNIP)	-	V X	X	X
Connected name identification presentation (CONP)	-	V X	X	X
Calling line identification presentation (CLIP)	V X	V X	X	X
Calling line identification restriction (CLIR)	V X	V X	X	X
Connected line identification presentation (COLP)	V X	V X	X	X
Connected line identification restriction (COLR)	V X	V X	X	X

Supplementary services and ISDN variants

ISDN supplementary services are implemented differently under various variants. NMS ISDN supports a subset of supplementary services in the following variants:

- The Digital Signaling System No. One (DSS1) European Telecommunications Standard (ETS) produced by the Signaling Protocols and Switching (SPS) Technical Committee of the European Telecommunications Standards Institute (ETSI). In this document, this variant is referred to as the ETS 300 variant.
- The Q.SIG signaling system for corporate networking. Q.SIG is standardized by a consortium of standards-producing bodies, including ETSI, the European Computer Manufacturer's Association (ECMA), and the International Organization for Standardization (ISO). The production effort is coordinated by the IPNS Forum. In this document, this variant is referred to as the Q.SIG variant.
- The National ISDN Protocol specified by Telcordia (previously Bellcore). The National ISDN protocol was developed to be a common, vendor neutral protocol for North America that would enhance interoperability of equipment by replacing several proprietary vendor protocols. In this document, this variant is referred to as the National ISDN 2 (NI2) variant.
- The ISDN Primary Rate User-Network Interface Specification by Nortel Networks. In this document, this variant is referred to as the DMS variant.

For other variant specification information, refer to the *Dialogic® NaturalAccess™ ISDN Software Installation Manual*.

ETS 300 variant specifications

The NMS ETS 300 variant implementation is a reference to the Digital Subscriber Signaling System No. One (DSS1) European Telecommunications Standard (ETS) produced by the Signaling Protocols and Switching (SPS) Technical Committee of the European Telecommunications Standards Institute (ETSI). The standard can be categorized as a basic Q.931 protocol with the addition of ASN.1 facilities to support ISDN supplementary services.

The following documents describe supplementary services as they are implemented in the NMS ETSI supplementary service package:

- ETS 300 102-1 (1990): "Integrated Services Digital Network (ISDN) User-network interface layer 3; Specifications for basic call control."
- ETS 300 195-1: "Integrated Services Digital Network (ISDN); Supplementary service interactions; Digital Subscriber Signaling System No. One (DSS1) protocol; Part 1: Protocol specification."
- ETS 300 196-1 (1993): "Integrated Services Digital Network (ISDN); Generic functional protocol for the support of supplementary services; Digital Subscriber Signaling System No. One (DSS1) protocol; Part 1: Protocol Specification."
- ETS 300 207-1 (1994): "Integrated Services Digital Network (ISDN); Diversion supplementary services; Digital Subscriber Signaling System No. One (DSS1) protocol; Part 1: Protocol Specification."

Q.SIG variant specifications

Q.SIG specifications are produced by a consortium of standards-producing bodies, including ETSI, the European Computer Manufacturer's Association (ECMA), and the International Organization for Standardization (ISO). The production effort is coordinated by the IPNS Forum.

The NMS ISDN Q.SIG implementation is based on ETSI second-edition specifications. These were produced after ECMA-created standards (based on CCITT standards and enhanced by ECMA after a European Commission mandate to produce such standards) were submitted to the ISO/IEC.

The following table contains a partial list of standards documents that are relevant to the NMS ISDN implementation. Where two standards are listed, the first is the stage 1/stage 2 standard and the second is the stage 3 (Q.SIG) standard.

Q.SIG service name	**ECMA standard and publication date**	**ETSI standard and publication date**	**ISO/IEC standard and publication date**
Basic call (64 kb/s unrestricted, 3.1 kHz audio and speech bearer services)	ECMA-142/143, December 2001	ETS300 171/172, June 2003	IS 11574/11572, 2000
Calling name identification presentation	ECMA 163/164, December 2001	ETS300 237/238, June 2003	IS 13864/13868, 1995
Connected name identification presentation	ECMA 163/164, December 2001	ETS300 237/238, June 2003	IS 13864/13868, 1995
Calling/connected name identification restriction	ECMA 163/164, March 1992	ETS300 237/238, June 1993	IS 13864/13868, 1995
Generic functional procedures	ECMA 165, June 2001	ETS300 239, June 2003	IS 11582, 2002
Call forwarding unconditional	ECMA 173/174, June 1992	ETS300 256/257, November 1993	IS 13872/13873 1995
Call forwarding busy	ECMA 173/174, June 1992	ETS300 256/257, November 1993	IS 13872/13873, 1995
Call forwarding no reply	ECMA 173/174, June 1992	ETS300 256/257, November 1993	IS 13872/13873, 1995
Call transfer	ECMA 177/178, December 2001	ETS300 260/261, August 2003	IS 13865/13869, 2003
Advice of charge, start of call	ECMA 211/212, December 1994		
Advice of charge, during call	ECMA 211/212, December 1994		
Advice of charge, end of call	ECMA 211/212, December 1994		
Calling line identification presentation	ECMA-148, June 1997	ETS300 173, May 1996. There is no stage 3 standard for this supplementary service and Q.SIG support is covered by the basic call stage 3 standard.	IS 14136, 1995. There is no stage 3 standard for this supplementary service and Q.SIG support is covered by the basic call stage 3 standard.

Q.SIG service name	ECMA standard and publication date	ETSI standard and publication date	ISO/IEC standard and publication date
Connection line identification presentation	ECMA-148, June 1990. There is no stage 3 standard for this supplementary service and Q.SIG support is covered by the basic call stage 3 standard.	ETS300 173, December 1992. There is no stage 3 standard for this supplementary service and Q.SIG support is covered by the basic call stage 3 standard.	IS 14136, 1995. There is no stage 3 standard for this supplementary service and Q.SIG support is covered by the basic call stage 3 standard.
Calling/connected line identification restriction	ECMA-148, June 1990. There is no stage 3 standard for this supplementary service and Q.SIG support is covered by the basic call stage 3 standard.	ETS300 173, December 1992. There is no stage 3 standard for this supplementary service. and Q.SIG is covered by the basic call stage 3 standard.	IS 14136, 1995. There is no stage 3 standard for this supplementary service and Q.SIG support is covered by the basic call stage 3 standard.

Q.SIG specifications

In the ISO specifications, Q.SIG is referred to as Private Signaling System No. One. The Q.SIG specifications identify different types of private ISDN network exchanges (PINXs) and different supplementary services. This creates a conformance matrix.

Q reference points

The Q reference point is different from typical reference points in that it describes the functions of a part of the network, rather than describing a point of interface to the network. The reference point location also implies that there is no user or network side of a connection. Rather, all Q.SIG signaling is symmetric between adjacent nodes.

NI2 variant

The NMS NI2 variant implementation is a reference to the National ISDN Protocol specified by Telcordia. The standard can be categorized as a basic Q.931 protocol with the addition of ASN.1 facilities to support NMS ISDN supplementary services.

The GR-2865-CORE, Issue2 (1997): "Two B Channel Transfer (TBCT) Bellcore Generic Requirements" document describes supplementary services as they are implemented in the NMS NI2 supplementary service package.

DMS variant

The NMS DMS variant implementation is based on the ISDN primary rate interface (PRI) user-network interface specification of Nortel Networks. The specification defines the interface between the Nortel Networks ISDN DMS-100 switch and user equipment.

Supplementary service participants

Three parties are identified in the activation or invocation of a supplementary service:

Participant	Description
Served user	The user or node that activates or invokes the supplementary service. If the service is billable, then this is the user or node charged for it.
Originating user	The user or node that originated the call into the network.
Diverted-to user	In call forwarding services, this is the user or node to which the call is rerouted.

Supplementary services under ETS 300

This section provides an overview of the supplementary services available with the ETS 300 variant, and how these services operate in the network architecture. It describes:

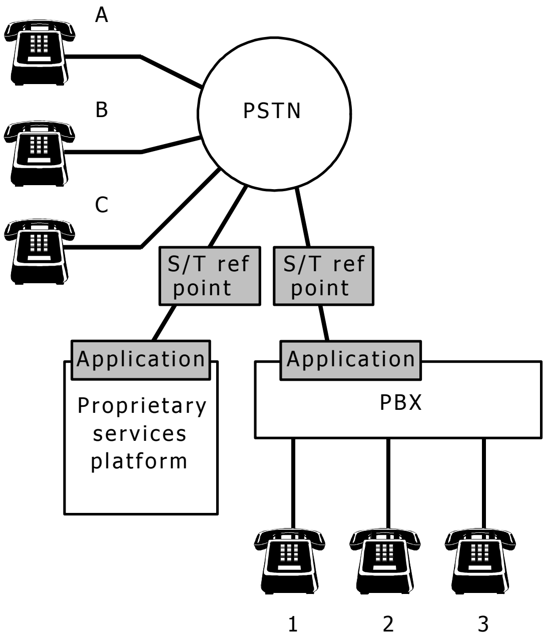
- ETS 300 variant
- Subscription and activation of supplementary services
- Hold and retrieve services
- Call transfer services
- Call forwarding services
- Advice of charge services
- Call identification services

ETS 300 variant

ETS 300 specifications describe a protocol designed to access network services from an intelligent user terminal. The protocol is not symmetric. It requires two distinct roles, the user side and the network side. NMS ISDN supports both sides for basic call control. Supplementary services, at this time, are implemented only for the user side.

The following illustration shows a sample network and the points where an NMS ISDN ETS 300 supplementary service application can interface with the network. As shown in the illustration, an NMS ISDN application written for the ETS 300 variant interfaces with the network on the user side of the S/T reference point of a CEPT E1 PRI ISDN trunk.

Network illustration showing position of ETSI application



KEY:

Application:	NMS ISDN application on user side PRI
A...C:	Users on PSTN
1 ... 3:	Users on local PBX served by user side PRI

Supplementary services over the T reference point are generally requests by the application for the network to perform an action on behalf of a subscriber or interface (for example, transfer a call).

Subscription and activation of supplementary services

Under ETS 300, most supplementary services require subscription. Subscription services are optional services provided by the network operator on a provisioning basis. When requesting network services from a provider, you can also request one or more subscription services. The services are not available unless the interface is provisioned with them.

Note: Some supplementary services may not require subscription. For example, the call hold service may be generally available.

Under ETS 300, most supplementary services must be activated before they are used. Activation is the process of turning on a service at the network or stack level. Once a service is activated, the network, the stack and/or the application can invoke (use) the service when needed.

A supplementary service can be activated in at least one of the following ways:

- Some supplementary services are activated the moment they are subscribed to. The remind diversion service is one of these. To deactivate this service, the service provider must be contacted.
- Other supplementary services are activated or deactivated by the application as necessary. The call diversion service is an example: the application can activate it to configure the network to automatically forward a user's calls.
- Many supplementary services can be set up either way. For example, when subscribing to advice of charge (AOC) services, the user can specify that the service is active at all times. Alternatively, services can be configured so they are active only when the application requests an activation.

The act of using an activated supplementary service is called invocation of the service. In some cases, an application can automatically activate an inactive supplementary service by invoking it: the activation and invocation occur simultaneously. The explicit call transfer service is an example: the application can invoke this service on a call-by-call basis.

Hold and retrieve services

An application on the user side of the S/T reference point may invoke hold and retrieve services on the network. When a call (identified by its connection ID) is placed on hold, the bearer channel (B channel) resource for the call is deallocated without losing the context of the call. The network side of the S/T reference point then reserves the B channel for allocation in a subsequent call offered by the user side.

The following notify hold and notify retrieve operations can be performed under ETS 300:

Operation	**Usage**
Notify hold	The network informs a party that it is on hold.
Notify retrieve	The network informs a held party that it has been retrieved.

For more information, see *Notify hold (ETS 300)* on page 53 and *Notify retrieve (ETS 300)* on page 55.

Call transfer services

Call transfer services allow an application at the user side of the S/T reference point to join two existing calls on the network side.

The following call transfer operations can be performed under ETS 300:

Operation	**Usage**
Invoke explicit call transfer	An application sends a request to the network to join two existing calls.
Notify transfer	The network notifies the joined users when a call has been affected by a remote transfer.

Call forwarding services

Under ETS 300, two types of call forwarding services are available: call diversion and call deflection.

Call diversion

Call diversion is activated by the served user application on the network for all calls for a specific user or trunk. With this service active, the network reroutes calls addressed to a specific user or trunk, without consulting the user side of the S/T reference point. Three types of call diversion are supported:

- Call forwarding - unconditional
- Call forwarding - busy
- Call forwarding - no response

The following call diversion operations can be performed under ETS 300:

Operation	Usage
Activate diversion	Activates call diversion on all calls on a user or trunk.
Deactivate diversion	Deactivates call diversion on a specific user or trunk.
Notify diversion	When a diversion or deflection occurs, the network notifies the diverted-to user or trunk of the rerouting operation.
Enquire diversion	The served user application can enquire the network, to learn the status of the call diversion service for a given user or trunk, or for all users/trunks.

Call deflection

Call deflection can be activated for all calls, or activated on a call-by-call basis. When invoked, the served user stack (not the network) deflects (redirects) the call to a new destination. With this service, the user side can deflect a call to a different destination, without first answering it.

The following call deflection operations can be performed under ETS 300:

Operation	Usage
Activate deflection	Activates call deflection for all calls on a specific trunk.
Deactivate deflection	Deactivates call deflection on a specific trunk.
Invoke deflection	Invokes call deflection on a specific call.

A special remind diversion service can also be activated (on a subscription basis) for an ETS 300 application. When a served user initiates an outbound call, the remind diversion service reminds the served user if call diversion has been activated for incoming calls.

Advice of charge services

Advice of charge (AOC) services provide the user with a way of tracking the costs of a specific call, in real time. Three separate AOC services are available, depending on when the application requires AOC information:

- AOC at start of call (AOC-S)
- AOC during the call (AOC-D)
- AOC at end of call (AOC-E)

The following advice of charge operations can be performed under ETS 300:

Operation	Usage
Advice of charge request	An application sends a request to the network to invoke advice of charge services for a specific call.
Advice of charge inform	Invoked by the network to pass advice of charge information to the application.

Call identification services

The following identification services are implemented in NMS ISDN for the ETS 300 variant:

Service	Usage
Calling line identification presentation (CLIP)	The called party receives the calling party's address information.
Calling line identification restriction (CLIR)	Prevents the calling party's address information from being presented to called users.
Connected line identification presentation (COLP)	Allows the calling party to determine the connected party's address information.
Connected line identification restriction (COLR)	Restricts the calling party from determining the connected party's address information.

Supplementary services under NI2

This section provides an overview of the supplementary services available with the NI2 variant and how these services operate in the network architecture. It describes:

- The National ISDN 2 (NI2) variant
- Using supplementary services
- PRI two B channel transfer
- Call identification services

National ISDN 2 (NI2) variant

NI2 specifications describe a protocol designed to access network services from an intelligent user terminal. The protocol is not symmetric, and requires two distinct roles, the user side and the network side. NMS ISDN supports both sides for basic call control, but NMS ISDN supplementary services only support the user side.

Using supplementary services

Under NI2, most supplementary services require subscription. Subscription services are optional services provided by the network operator by request. When you request network services from a provider, you can also request one or more subscription services. These services are not available unless the interface is provided with them.

Note: Some supplementary services may not require subscription.

PRI two B channel transfer

The two B channel transfer (TBCT) service allows an application at the user side to request the transfer of two independent calls. To use the service, the user should be subscribed to TBCT.

The following table lists call transfer operations that can be performed under NI2:

Operation	Description
Invoke two B channel transfer	The application sends a request to the network to transfer two independent calls.
Two B channel transfer notification	The network notifies the application that the call previously transferred by TBCT has been cleared. To receive this notification, the user should be subscribed to the notification to controller feature.

Call identification services

All of the call identification services for the ETS 300 variant are also implemented for NI2. In addition, the following identification services are implemented in NMS ISDN:

Service	Usage
Calling name identification presentation (CNIP)	The called party receives the name of the calling party. Available only under the NI2 and Q.SIG variants.
Connected name identification presentation (CONP)	The calling party receives the name of the called party. Available only under the NI2 and Q.SIG variants.

Supplementary services under Q.SIG

This section provides an overview of the supplementary services available with the Q.SIG variant and how these services are implemented in the network architecture. It describes:

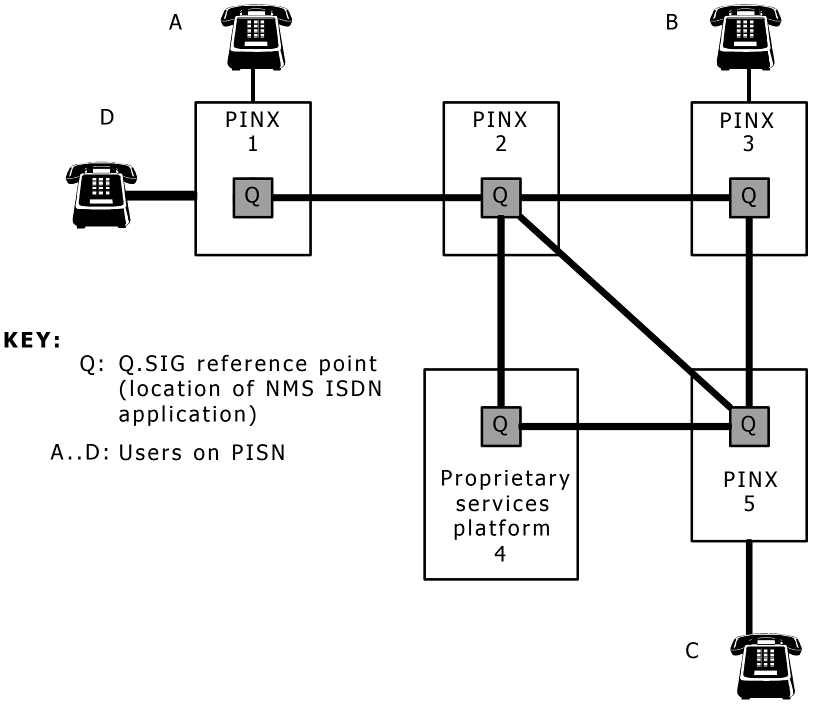
- The Q.SIG variant
- Using supplementary services
- Tandem services
- Transfer services
- Call forwarding services
- Call identification services

Q.SIG variant

Q.SIG specifications describe interactions between nodes in a private ISDN network (PISN). Each node is a private ISDN network exchange (PINX). These nodes play identical roles in the network. Q.SIG is a symmetric protocol because there is no distinct user side and network side.

Network illustration showing position of Q.SIG application

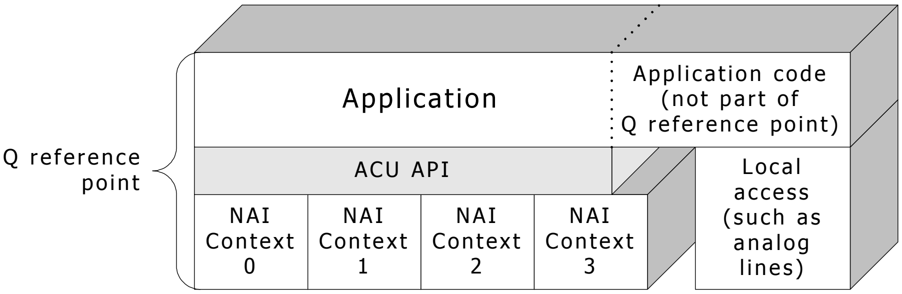
The following illustration shows a sample Q.SIG network and the reference points where an NMS ISDN Q.SIG supplementary service application can join the network. As shown in the illustration, an NMS ISDN application written for this variant interfaces with the network at the Q reference point. The role of the application is to implement call control, and to control message exchanges at the reference point.


A
B
D
PINX 1
PINX 2
PINX 3
Q
Q
Q
KEY:
Q: Q.SIG reference point (location of NMS ISDN application)
A..D: Users on PISN
Q
Q
Proprietary services platform 4
PINX 5
C

Q reference point

The following illustration shows the part of an NMS ISDN application that is the Q reference point.

Note: Application code related to local access (such as analog lines) is not considered part of the Q reference point.



Supplementary services over the Q reference point are generally requests sent by the application for services from another node in the network, or notifications that the local node performed various services.

Note: If you are building a Q.SIG application, the PINX node address must be specified in your initial call to **isdnStartProtocol**. For more information, see *Specifying the Q.SIG node address* on page 47.

Using supplementary services

Q.SIG applications provide supplementary services as part of their basic duties: the subscription concept has no meaning at this level. All supported services are activated at all times. A Q.SIG application only needs to invoke a service to use it.

Tandem services

Tandem services support the transit node role. A transit node is an intermediate step in a call being set up through a network. A PINX may take on the responsibilities of a transit node during the call setup procedures as a result of routing decisions, or it may happen as a result of supplementary service activation (such as call transfer). A transit node must maintain two separate basic calls, and combine the events from one call with actions on the other call.

NMS ISDN supports the call bridging tandem service. When this service is active, all notification and facility information elements are passed from one end to the other through the transit node. However, the application remains responsible for basic call control interaction. For example, the application must handle hanging up.

The call bridging service can be invoked in either of the following two ways:

- Explicitly, using the invoke bridge calls operation.
- Implicitly when the notify transfer operation is called, and both transfer parties are remote. Implicit call bridging does not take place if one or both calls are local.

Transfer services

NMS ISDN supports transfer-by-join operations between Q.SIG nodes. In a transfer-by-join operation, two separate calls are connected through the local node. The local node is still involved in the call (the call is not rerouted).

To transfer a call under Q.SIG, the application must perform the switching required to connect calls together, using standard calls to the Natural Access Switching service. The application also invokes a notify transfer operation to notify each remote party that the transfer took place.

Call forwarding services

Under Q.SIG, call diversion supports:

- Call forwarding - unconditional
- Call forwarding - busy
- Call forwarding - no response

The following call diversion operations are supported under Q.SIG:

Operation	Usage
Invoke call diversion	The served user node sends a request to the originating node to forward a call.
Notify diversion	The originating node notifies the diverted-to node that the call has been forwarded.

Call identification services

All of the call identification services for the ETS 300 variant are also implemented for Q.SIG. In addition, the following Q.SIG-only identification services are implemented in NMS ISDN:

Service	Usage
Calling name identification presentation (CNIP)	The called party receives the name of the calling party. Available only under the Q.SIG variant.
Connected name identification presentation (CONP)	The calling party receives the name of the called party. Available only under the Q.SIG variant.

Supplementary services under DMS

This section provides an overview of the supplementary services available with the DMS variant and how these services operate in the network architecture. It describes:

- DMS variant
- Using supplementary services
- Release link trunk (RLT) service

DMS variant

DMS specifications describe a protocol designed to access network services from an intelligent user terminal. The protocol is not symmetric, and requires two distinct roles, the user side and the network side. NMS ISDN supports both sides for basic call control, but NMS ISDN supplementary services only support the user side.

Using supplementary services

Under DMS, most supplementary services require subscription. Subscription services are optional services provided by the network operator on request. When you request network services from a provider, you can also request one or more subscription services. These services are not available unless the interface is provided with them.

Release link trunk (RLT) service

The release link trunk (RLT) service allows an application at the user side to request the transfer of two independent calls. To use the service, the user should be subscribed to RLT.

The following table describes the call transfer operation that can be performed under DMS:

Operation	**Description**
Invoke two B channel transfer	The application sends a request to the network to transfer two independent calls.

4 Programming model

Supplementary services in ACU messages

The NMS ISDN supplementary services are modeled as a protocol element separate from the call control protocol element. The call control element of a message is used to manage the basic call state, while the supplementary services element of a primitive manages the supplementary service states.

In order for these two elements to work together but retain separation, the NMS ISDN basic call control buffer format allows for supplementary service information to be carried with the basic call control primitives in separate structures. In most cases, the structures are attached to ACU_FACILITY_RQ or ACU_FACILITY_IN messages as explicit control signaling for supplementary services. However, there are cases where extended data structures containing supplementary service information are attached to basic call control primitives such as ACU_CONN_RQ.

A single ACU message can carry multiple supplementary service structures. Each supplementary service structure contains information pertaining to a specific supplementary service.

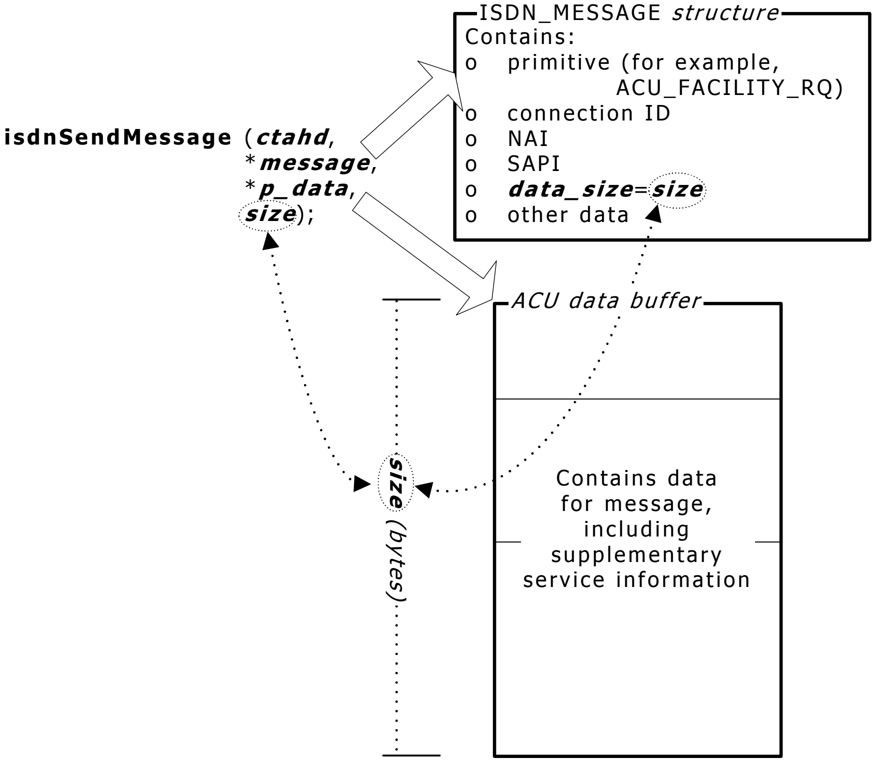
Components of ACU messages

To send a message to the NMS ISDN protocol stack, the application builds two main structures and then calls **isdnSendMessage** in the NMS ISDN Messaging library. The structures specified in the function call are:

- ISDN_MESSAGE. In this structure, the application specifies the message to be sent using one of the message primitives documented in the *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual*. Also specified is information identifying the context of the message, such as the connection ID, the NAI, and the SAPI.
- A data buffer for messages that require additional data. The data (if any) differs for each message type. If supplementary service information is sent, it is included in this data.

The application assigns values to the fields in the ACU message structure using macros. These macros are defined in *isdnacu.h*. See the *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual* for more information about message primitives and macros.

The following illustration shows the content and meaning of each of the arguments sent in an **isdnSendMessage** function call:

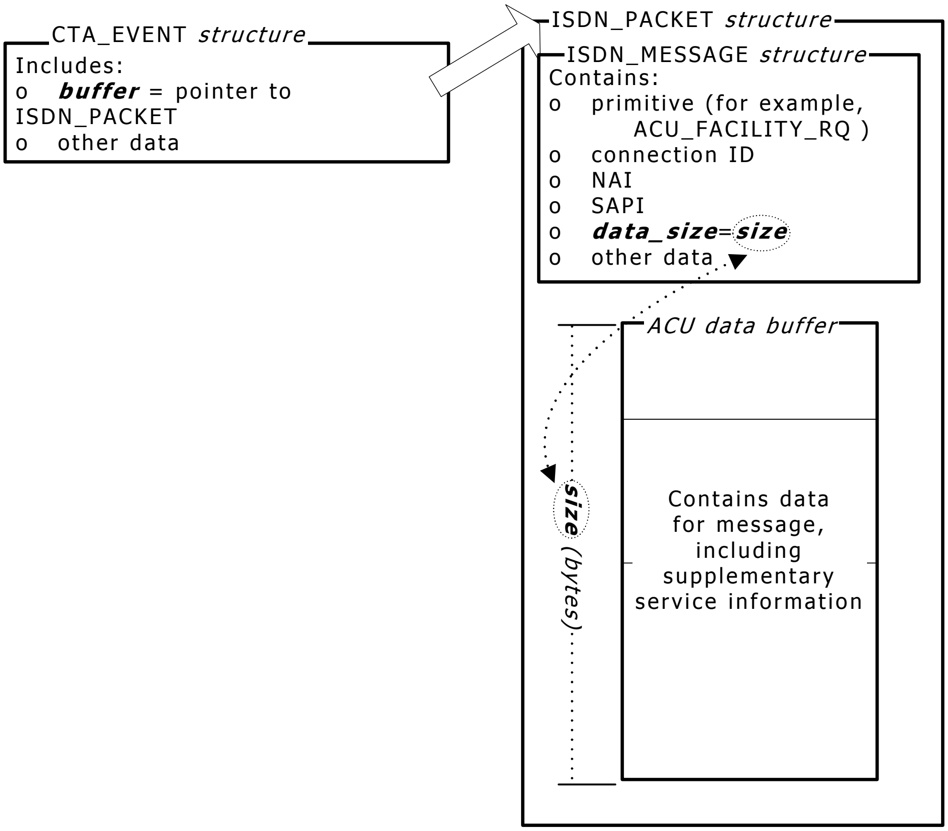


When the ACU data structures reach the ISDN protocol stack, the stack rearranges the data into several Q.931 information elements (IEs), builds a complete Q.931 message with the IEs, and sends it to the network.

When an NMS ISDN protocol stack message is received, an ISDNEVN_RCV_MESSAGE event occurs, using the standard Natural Access event handling mechanism. ***buffer*** in the Natural Access event structure is a pointer to an ISDN_PACKET structure. This structure contains:

- An ISDN_MESSAGE structure containing the message and other data.
- A buffer containing the message. If supplementary service information is received, it is included in this data.

The following illustration shows the structure of this message packet:

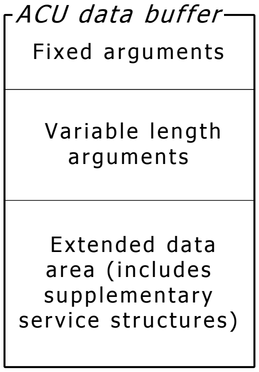

CTA_EVENT *structure*
Includes:
o ***buffer*** = pointer to ISDN_PACKET
o other data
ISDN_PACKET *structure*
ISDN_MESSAGE *structure*
Contains:
o primitive (for example, ACU_FACILITY_RQ)
o connection ID
o NAI
o SAPI
o ***data_size***=***size***
o other data
ACU data buffer
Contains data for message, including supplementary service information
size *(bytes)*

Components of the ACU data buffer

The ACU data buffer sent or received from the stack is made up of several blocks of information including:

Information block	**Description**
Fixed argument area	This is a C structure that is specific to the message primitive. It contains fixed length information required to process the primitive. It also contains offsets to fields in the variable length argument area. All information in this area is specified using macros specific to the primitive. (See the *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual* for more information.)
Variable length argument area	This area contains variable length fields referenced by offsets in the fixed length structure. Data such as the called digit string are stored here. (See the *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual* for more information.)
Extended data area	This area can be used for a variety of purposes. This manual describes how this area is used to send and receive supplementary service information.

The following illustration shows the components of an ACU data buffer:


ACU data buffer
Fixed arguments
Variable length arguments
Extended data area (includes supplementary service structures)

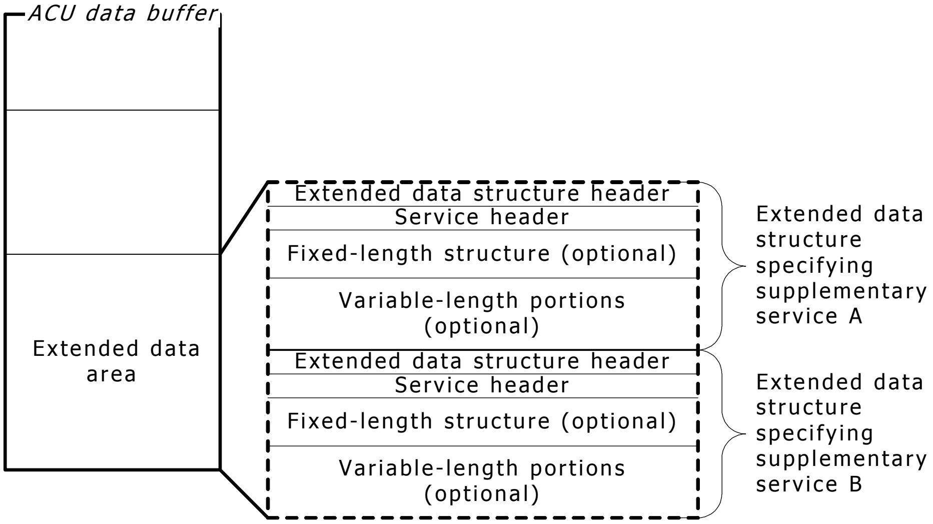
Components of the extended data area

Within the extended data area, one or more supplementary service structures appear, one after the other. Each structure contains:

- An extended data structure header, identifying the structure as a service structure.
- A header identifying the type of service, and whether it is an invocation, rejection, or positive or negative acknowledgement.
- (Optional) A fixed length structure containing data specific to the supplementary service type and operation. The structures for each supplementary service can be found in *isdnacu.h*. Each structure is a standard C structure that can be accessed through a pointer cast to the appropriate type.
- (Optional) Variable length information pointed to in the fixed length structure.

In some cases, there is no additional data, so there is no fixed structure or variable length information beyond the header.

The following illustration shows the extended data area of an ACU message buffer:


ACU data buffer
Extended data area
Extended data structure header
Service header
Fixed-length structure (optional)
Variable-length portions (optional)
Extended data structure specifying supplementary service A
Extended data structure header
Service header
Fixed-length structure (optional)
Variable-length portions (optional)
Extended data structure specifying supplementary service B

Specifying supplementary services

To specify supplementary services, the application performs the following tasks:

1. It finds the beginning of the extended data area of the ACU message buffer.
2. In this area, it fills the structure for a supplementary service.
3. It finds the end of the structure just filled, adds another supplementary service substructure if necessary, and so on.

Most of these tasks are performed using specific macros and pointers. These tasks are listed in the following sections. A sample code fragment showing how to create a supplementary service specification is also included.

Initializing the extended data area

To begin specifying supplementary services, the application finds the end of the fixed and variable-length argument areas of the ACU message buffer, and sets a specific pointer there. The following table lists the pointers and macros used for these tasks:

Macro	Description
Acu_ext_descr_offset	Must be sent to indicate the beginning of the extended data area of the ACU message buffer. Assumes that a pointer, p_data, points to the first data character of the ACU message buffer.
Acu_***xxxx_yy***_start_ext_data	Indicates the offset to the first byte of the extended data area for message ***xxxx_yy***. ***xxxx_yy*** is the name of an ACU message primitive minus the ACU_ prefix, in lowercase such as conn_rq and facility_rq. For example: Acu_conn_rq_start_ext_data
p_ext_data	Pointer to the beginning of the extended data structure currently being filled.

To initialize the extended data area:

Step	Action
1	The application completes the construction of the fixed and variable length argument areas of the ACU data buffer. The *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual* describes how to do this. The extended data area can be filled only when the rest of the ACU data buffer has been completed. If you are building a Q.SIG application, the PINX node address is specified in your initial call to **isdnStartProtocol**. For more information, see *Specifying the Q.SIG node address* on page 47.
2	The application declares the p_ext_data pointer. This pointer will indicate the beginning of the extended data structure currently being filled. `unsigned char *p_ext_data;`
3	The application sets the Acu_facility_code macro to ACU_FAC_EXT_SERVICE to indicate that the extended data area is to be used (if the supplementary service specification is to be sent with an ACU_FACILITY_RQ primitive).This macro does not need to be set if any other ACU primitive is used. `Acu_facility_code = ACU_FAC_EXT_SERVICE;`
4	The application sets the Acu_ext_descr_offset macro to indicate the beginning of the extended data area of the buffer. Each fixed data structure contains a macro that specifies the start of the extended data area. `Acu_ext_descr_offset = Acu_facility_rq_start_ext_data;` **Note:** These macros assume that a pointer, p_data, points to the first character of the buffer.

When these tasks are complete, the application can begin filling extended data structures and adding them to this area, one after the other.

Filling extended data structures

To fill in an extended data structure for a supplementary service, the application first fills in the structure's service header, identifying the structure as a supplementary service structure and specifying the type and operation of the service. It then fills the data structure for the service.

If additional data structures are required (for example, the application is specifying more than one supplementary service in the message), the pointer p_ext_data is set to the end of the structure just filled, and a new one is added. The macro Acu_ext_ss_build_end facilitates this procedure.

The following table lists the macros used for these tasks:

Macro	Description
Acu_ext_ss_build_begin(***opid***, ***optype***)	Initializes a new supplementary service data structure. Sets p_ext_data to the beginning of the extended data structure, and stores the service operation type and id in the service structure header. ***opid*** is one of the operation IDs listed in Operation ID, and ***optype*** is one of the operation type identifiers listed in Operation type identifier.

Macro	Description
Acu_ext_ss_build_end (p_end)	Called to indicate that building of an extended data structure is complete. p_end is a pointer to the byte after the last byte of the current extended data structure. If this structure includes variable-length data, p_end points to the first byte after the last data element. This macro calculates the length of the extended data area and of the new supplementary service structure and stores these values in the header. It also counts the number of supplementary service structures in this specification and stores this value in the header.

To fill an extended data structure for a supplementary service:

Step	Action
1	The application calls the macro Acu_ext_ss_build_begin(***opid***, ***optype***) to initialize a new supplementary service data structure. (***opid*** is one of the operation IDs listed in Operation ID and ***optype*** is one of the operation type identifiers listed in Operation type identifier.) `Acu_ext_ss_build_begin(ACU_OP_ID_DEFLECTION,ACU_OP_TYPE_INVOKE);` The macro sets p_ext_data to the beginning of the extended data structure, and stores the service operation type and ID in the service structure header.

2	The application sets a pointer to the appropriate data structure so it equals p_ext_data. For example, for a call deflection invocation operation, the application uses the acu_ss_deflect_invoke data structure (defined in *isdnacu.h*):
	`struct acu_ss_deflect_invoke`
	`{`
	`struct acu_ext_hdr        ext_hdr;           /*Extension header     */`
	`struct acu_ss_hdr         ss_hdr;            /*Supp. services header*/`
	`struct acu_address        deflect_to;        /*No. to direct call to*/`
	`struct acu_ss_association charge_association;/*Optional, used when  */`
	`};                                  /*AOC-E service has been invoked*/`
	The following line shows how the application sets a pointer to this data structure:
	`DeflectInvoke = (struct acu_ss_deflect_invoke *)p_ext_data;`
	Note: If any optional strings are added, p_end must be adjusted for that string length.

3	The application now fills the extended data structure for the service. The extended data structure for each operation ID/operation type combination is listed in Supplementary service extended data structures.
4	The application calculates the end of the supplementary services data structure. `p_end += sizeof(struct acu_ss_deflect_invoke);`
5	The application calls Acu_ext_ss_build_end to indicate that the data structure is built: `Acu_ext_ss_build_end (p_end);` p_end is a pointer to the byte after the last byte of the current extended data structure. If this structure includes variable-length data, p_end points to the first byte after the last data element. This macro calculates the length of the extended data area and of the new supplementary service structure and stores these values in the header. It also counts the number of supplementary service structures in this specification, and stores this value in the header.
6	If there are additional supplementary service data structures to be added, steps 1 through 5 are repeated for each one.

Supplementary service specification code sample

The following code fragment shows how to create a supplementary service specification:

```
char *number;
unsigned char *p_end;
unsigned char *p_ext_data;
unsigned char JoinedConnID;
struct acu_facility *p_data;
struct acu_ss_notify_transfer_invoke *NotifyTransfer;

/*  Fill in the fixed portion of the facility message */
p_data = (struct acu_facility *)MsgBuffer;           /* Initialize p_data */

/*  First zero out the entire buffer */
memset(p_data, OFF, ISDN_BUFFER_DATA_LGTH);         /* Zero the structure */

/*  Tell the stack we are using supplementary services. Needed only if    */
                           /*  supp. service is sent in ACU_FACILITY_RQ.  */
Acu_facility_code = ACU_FAC_EXT_SERVICE;                 /* Supp. service */

/*  There is no more to do in the fixed structure except fill in header */
Acu_ext_descr_offset = Acu_facility_start_ext_data;

/*  Start supplementary service extended data structure, to invoke */
/*  Notify Transfer supplementary service */
Acu_ext_ss_build_begin(ACU_OP_ID_NOTIFY_TRANSFER,ACU_OP_TYPE_INVOKE);

/*  Set the structure pointer... */
NotifyTransfer = (struct acu_ss_notify_transfer_invoke *)p_ext_data;

/*  Calculate the address of where the data will go... */
p_end += sizeof(struct acu_ss_notify_transfer_invoke);

/*  FILL IN THE EXTENDED DATA STRUCTURE... */

/*  Some data fields follow */
NotifyTransfer->charge_id.invoke = 0;
NotifyTransfer->call_status = ACU_SS_CALL_STATUS_ANSWERED;
NotifyTransfer->end_designation = ACU_SS_END_DESIGNATION_PRIMARY;
/*  Redirecting number information */
NotifyTransfer->redir_nb.invoke = 1;
NotifyTransfer->redir_nb.presentation_restricted = 0;
NotifyTransfer->redir_nb.number_plan =
ACU_SS_NUMBER_PLAN_PRIVATE_LEVEL_1_REGIONAL;
NotifyTransfer->redir_nb.screen_ind = ACU_SS_SCREEN_USER_PROV_VERIFIED;

/*  Copy in a number to the data area */
number = "1234567";                                        /* Dummy data */

/*Calculate offset, store length, store the data, advance the pointer */
NotifyTransfer->redir_nb.offset =
        (unsigned char)(p_end - (unsigned char*)&NotifyTransfer->redir_nb);
NotifyTransfer->redir_nb.len = strlen(number);         /* Length of data */
strcpy((char *)p_end, number);                     /* Copy extended data */
p_end += strlen(number);
NotifyTransfer->joined_conn_id.board = BoardNumber;
NotifyTransfer->joined_conn_id.nai =  NAI;
NotifyTransfer->joined_conn_id.conn_num =  JoinedConnID;

/*  Field not used in this direction... */
NotifyTransfer->response_rq = 0;
```

```
/*  No sub-address information. */
NotifyTransfer->redir_sub.invoke = 0;
NotifyTransfer->redir_sub.pad = 0;
NotifyTransfer->redir_sub.type = 0;
NotifyTransfer->redir_sub.odd_even_ind = 0;
NotifyTransfer->redir_sub.offset = 0;
NotifyTransfer->redir_sub.len = 0;

/* Finished with this supp. service */
Acu_ext_ss_build_end(p_end)
```

Retrieving supplementary service information

To access supplementary service information, the application does the following:

Step	Action
1	It finds the beginning of the extended data area of the ACU message buffer, and determines whether there are any extended data structures in it.
2	If extended data structures are present, it reads the extended data structure header of the first extended data structure to determine if it is a supplementary service structure or not.
3	If the structure is a supplementary service structure, the application reads the operation type and ID in the service header to determine what data to expect.
4	It sets a pointer to the first byte of the data, and reads it in.
5	It reads the size of the structure from the extended data structure header, and moves the pointer.
6	It repeats steps 2 through 5 for each extended data structure in the ACU message buffer.

Most of these tasks are performed using specific macros and pointers, listed in the following table:

Macro	Description
Acu_ext_id	Identifies the type of an extended data structure in the structure's extended data structure header. ACU_EXT_SERVICE indicates a supplementary service data structure.
Acu_ext_ss_op_id	Identifies the operation ID of a supplementary service in the service header of the extended data structure specifying the service.
Acu_ext_ss_op_type	Identifies the operation type of a supplementary service in the service header of the extended data structure specifying the service.
Acu_ext_descr_nb	Indicates the number of extended data structures in the extended data area.
Acu_ext_descr_first_address	Indicates the address of the first extended data structure. Assumes that a pointer, p_data, points to the first data character of the ACU message buffer.
Acu_ext_lgth	Indicates the length of the data area of the current extended data structure (the fixed-length area plus the variable-length area, if any).

Identifying extended data structures

To identify supplementary service extended data structures in the ACU message buffer, the application:

Step	Action
1	Checks the macro Acu_ext_descr_nb to determine how many (if any) extended data structures there are in the buffer.
2	Sets p_ext_data to the first data character of the extended data structure buffer. `p_ext_data = Acu_ext_descr_first_address;`
3	Determines whether or not an extended data structure is a supplementary service structure by checking the macro Acu_ext_id. Its value (stored in the extended data structure header of the structure) should be ACU_EXT_SERVICES.

Reading a supplementary service extended data structure

To read a supplementary service extended data structure, the application:

Step	Action
1	Reads the values of Acu_ext_ss_op_type and Acu_ext_ss_op_id to determine the operation ID and type of the supplementary service data structure. This determines which data structure to expect. For example, if Acu_ext_op_id is ACU_OP_ID_NOTIFY_TRANSFER and Acu_ext_op_type is ACU_OP_TYPE_INVOKE, the data structure is acu_ss_notify_transfer_invoke.
2	Uses p_ext_data to pick up the address of the extended structure.
3	Reads the information in the structure.
4	Resets p_ext_data to the end of the structure: `p_ext_data = Acu_ext_descr_next_address;`
5	Repeats these steps Acu_ext_descr_nb times.

Supplementary service retrieval code sample

The following code fragment shows how to retrieve supplementary service information from an ACU message:

```
unsigned char *p_ext_data;
if (Acu_ext_descr_nb > 0)
  {
    p_ext_data = Acu_ext_descr_first_address;
    /* process extended parameters */
    for (i = Acu_ext_descr_nb; i > 0; i--)
    {
      process_acu_ext_element (p_ext_data);
      p_ext_data = Acu_ext_descr_next_address;
    }
 }
 void process_acu_ext_element(unsigned char *p_ext_data)
  {
    if (Acu_ext_id == ACU_EXT_SERVICES)
    {
      /* We have a parameter containing a service structure*/
      ....
      ProcessAcuExtService (p_ext_data)
    }
  }
  void ProcessAcuExtService(unsigned char *p_ext_data)
  {
      ushort op_id = Acu_ext_ss_op_id;
      ushort op_type = Acu_ext_ss_op_type;
      ulong event = (op_id << 16) + op_type;
      struct acu_ss_notify_transfer_invoke *Transfer;
      struct acu_ss_aoc_inform_invoke *AOC;
      switch(event)
      {
        case (ACU_OP_ID_NOTIFY_TRANSFER << 16) + ACU_OP_TYPE_INVOKE:
        {
         Transfer = (struct acu_ss_notify_transfer_invoke *)p_ext_data;
         ...
        }
        break;
        case (ACU_OP_ID_AOC_INFORM << 16) + ACU_OP_TYPE_INVOKE:
        {
         AOC = (struct acu_ss_aoc_inform_invoke *)p_ext_data;
         ...
        }
        break;
      }
  }
```

Supplementary service extended data structures

Each supplementary service structure contains:

- A unique operation ID stored in the service header.
- An operation type identifier stored in the service header.
- Data relevant to the operation. Each extended data structure is unique to the operation ID / operation type combination.

The structures for each valid combination can be found in *isdnacu.h*. Each structure is a standard C structure that can be accessed through a pointer cast to the appropriate type.

The following table lists the extended data structures used to specify supplementary services in the extended data area of an ACU message.

Various substructures are used to contain data for fields in these structures. For more information, see *Extended data structure substructures overview* on page 46.

Many fields in these structures are filled using predefined constants. For a list of the constants available for a field, refer to the Constants for isdnacu.h section of this manual.

Extended data structure	**Purpose**
acu_ss_act_divert_invoke	Request activation of call diversion service
acu_ss_act_divert_ret_error	Error in call diversion service activation request
acu_ss_act_divert_ret_result	Call diversion service activation request successful
acu_ss_activate_deflect_invoke	Request activation of call deflection service
acu_ss_activate_deflect_ret_result	Call deflection service activation request successful
acu_ss_aoc_inform_invoke	Information returned by advice of charge service
acu_ss_aoc_request_invoke	Request activation of advice of charge service
acu_ss_aoc_request_ret_error	Error in advice of charge service activation request
acu_ss_aoc_request_ret_result	Advice of charge service activation request successful
acu_ss_bridge_calls_invoke	Request invocation of bridge calls service
acu_ss_bridge_calls_ret_result	Bridge calls service invocation successful
acu_ss_deact_divert_invoke	Request deactivation of call diversion service
acu_ss_deact_divert_ret_error	Error in call diversion service deactivation request
acu_ss_deact_divert_ret_result	Call diversion service deactivation request successful
acu_ss_deactivate_deflect_invoke	Request deactivation of call deflection service
acu_ss_deactivate_deflect_ret_result	Call deflection service deactivation request successful
acu_ss_deflect_invoke	Request to invoke call deflection for a call
acu_ss_deflect_ret_error	Error in call deflection attempt
acu_ss_deflect_ret_result	Successful call deflection attempt
acu_ss_divert_invoke	Request to forward a call (call diversion)
acu_ss_divert_ret_error	Successful call diversion

Extended data structure	Purpose
acu_ss_divert_ret_result	Error in call diversion attempt
acu_ss_enquire_divert_invoke	Attempt to invoke enquire diversion (enquire the network for users' call diversion service status)
acu_ss_enquire_divert_ret_error	Error in enquire diversion invocation
acu_ss_enquire_divert_ret_result	Successful invocation of enquire diversion service
acu_ss_notify_diversion_invoke	Invoke notify diversion service
acu_ss_notify_diversion_ret_result	Successful notify diversion
acu_ss_notify_hold_invoke	Invoke notify hold service
acu_ss_notify_retrieve_invoke	Invoke notify retrieve service
acu_ss_notify_tbct_calls_ret_result	Successful notify two B channel transfer
acu_ss_notify_transfer_invoke	Invoke notify transfer service
acu_ss_notify_transfer_ret_result	Successful notify transfer invocation
acu_ss_reject	Generic rejection message
acu_ss_reminder_diversion_invoke	Invoke remind diversion service
acu_ss_retrieve_invoke	Invoke call retrieve service
acu_ss_retrieve_ret_result	Successful call retrieve service invocation

Operation ID

An operation ID is included in each supplementary service operation call. The ID indicates the supplementary service operation to which the structure pertains.

Note: Identification supplementary services (CLIP, CNIP, COLP, COLR, CONP, and CLIR) do not have operation IDs. They are accessed using fields in the basic call control primitives.

The following table lists valid operation IDs:

Supplementary service operation	**Operation ID**
Invoke bridge calls	ACU_OP_ID_BRIDGE_CALLS
Notify hold	ACU_OP_ID_NOTIFY_HOLD
Notify retrieve	ACU_OP_ID_NOTIFY_RETRIEVE
Explicit call transfer	None
Notify transfer (Q.SIG and DPNSS)	ACU_OP_ID_NOTIFY_TRANSFER
Invoke two B channel transfer	None
Notify two B channel transfer	None
Invoke call diversion	ACU_OP_ID_DIVERSION
Activate diversion	ACU_OP_ID_ACTIVATE_DIVERSION
Deactivate diversion	ACU_OP_ID_DEACTIVATE_DIVERSION
Enquire diversion	ACU_OP_ID_ENQUIRE_DIVERSION
Remind diversion	ACU_OP_ID_REMIND_DIVERSION
Notify diversion (Q.SIG and ETS 300)	ACU_OP_ID_NOTIFY_DIVERSION
Invoke call deflection	ACU_OP_ID_DEFLECTION
Activate deflection	ACU_OP_ID_ACTIVATE_DEFLECTION
Deactivate deflection	ACU_OP_ID_DEACTIVATE_DEFLECTION
Advice of charge request	ACU_OP_ID_AOC_REQUEST
Advice of charge inform	ACU_OP_ID_AOC_INFORM
Calling name identification presentation (CNIP)	None
Connected name identification presentation (CONP)	None
Calling line identification presentation (CLIP)	None
Calling line identification restriction (CLIR)	None
Connected line identification presentation (COLP)	None
Connected line identification restriction (COLR)	None

Operation type identifier

An operation type identifier (optype) is included in each supplementary service operation call. The optype indicates what the supplementary service message means: why the message is being sent or received.

There are four valid operation types:

Operation type	**Meaning**
ACU_OP_TYPE_INVOKE	The message is a request to invoke the supplementary service operation.
ACU_OP_TYPE_REJECT	The message indicates rejection of a request to invoke the operation. (See the *acu_ss_reject* on page 160 extended data structure.)
Return result (ACU_OP_TYPE_RETRES)	The message indicates successful completion of the operation.

Operation type	Meaning
Return error (ACU_OP_TYPE_RETERR)	The message indicates unsuccessful completion of the operation.

ACU primitives and supplementary services

The relationship between the ACU primitive type and the supplementary service operation that is carried can be divided into the following categories:

- Tightly coupled services
- Loosely coupled services
- Connection-independent services

Tightly coupled services

Tightly coupled services and primitives require that the supplementary service operation identifier and operation type be associated with a specific ACU primitive. For example, invocation of the advice of charge request supplementary service must always be done within an ACU_CONN_RQ primitive.

When a service is tightly coupled with an ACU primitive, the service description in this document lists the primitive to use. If the application sends the service request in the wrong primitive, the stack returns ACU_OP_TYPE_REJECT with a local cause of ACU_SS_REJECT_LOCAL_INVALID_STATE in response.

Loosely coupled services

Loosely coupled services and primitives share the same connection ID. In other words, the service applies to the same connection as the basic primitive. However, these services can be carried in any type of basic primitive.

When a loosely coupled service structure must be sent over the interface and a basic call control primitive is also being sent, the two can be combined into a single buffer. If there is no basic call primitive, the ACU_FACILITY_RQ/IN/CO primitive can be used as a NULL basic call message and the service structure can be attached to it. For example, invocation of the Advice of Charge Inform service can be carried in any of the basic call primitives (ACU_ALERTING_IN, ACU_CLEAR_IN), or it may be in an ACU_FACILITY_IN.

Even though the primitive and the service are loosely coupled, the call state may prohibit the successful invocation of the service. For example, the explicit call transfer (ECT) service cannot be invoked on an inbound call that has not been answered. However, the service request can be sent in the ACU_CONN_RS primitive or a subsequent ACU_FACILITY_IN primitive.

Most supplementary services are loosely coupled with the primitive that carries them. The application must not assume which primitive type will carry a loosely coupled service.

Connection-independent services

Connection-independent services are not associated with a call in any way. An example of a connection-independent service is the activate diversion supplementary service.

Because there is no connection number associated with this service, a different mechanism is used to allow the discrimination between connection-oriented and connection-independent signaling. The sapi field is used for this purpose.

The sapi field of a connection-oriented message (for example, ACU_CONN_RQ) is set to ACU_SAPI. The sapi field of a connection-independent message is set to ACU_SAPI_MGT. The ACU_FACILITY_RQ/IN/CO are the only primitives used to carry connection-independent service structures.

When the application constructs a connection-independent service structure, it should attach it to an ACU_FACILITY_RQ and set the value of the sapi field to ACU_SAPI_MGT. The application's discrimination function must not use the connection ID field of the message when the sapi field indicates a connection-independent message.

Extended data structure substructures overview

Various substructures are referenced in the supplementary service extended data structures. These substructures contain particular types of information, such as address or subaddress information. For a list of the data structures, refer to *Supplementary service extended data structures* on page 42.

Each of these structures contains an invoke field used to indicate the presence or absence of useful data in the structure. If the field is set to ON, the stack uses the information in the structure. Otherwise, the information in the structure is ignored. If the successful invocation of the service requires the inclusion of the parameter and the invoke field is OFF, the service request is rejected.

Some structures contain a reference to a string location and string length. The string location is specified using the offset field in the structure. This offset value is calculated from the address of the structure containing the offset field.

The len field in the structure contains the length of the data. If you null-terminate the field, the length field must not include the null terminator.

Note: Structures sent from the stack do not include null terminations.

If a stack or network error occurs when an attempt is made to invoke or activate a service, the acu_ss_reject substructure is returned in an ACU indication or confirmation message. The fields in this substructure indicate the cause of the rejection.

The following table describes each of the extended data substructures:

Substructure	Description
acu_address	Specifies the full address and subaddress of a party.
acu_call_ret	Specifies a call reference of a party.
acu_conn_id	Specifies the connection information of a party.
acu_party_name	Specifies the name information of a party.
acu_party_num	Specifies the presentation indicator, numbering plan, and screen indicator of a party.
acu_party_subaddress	Specifies the subaddress of a party.
acu_ss_association	Contains a charge ID and charged number, for tracking charging information on transferred or forwarded calls. See *AOC and explicit call transfer (ECT) services* on page 103 and *AOC and call deflection services* on page 104.
acu_ss_chan	Specifies the channel information for a party.

Specifying the Q.SIG node address

If you are building a Q.SIG application using supplementary services, you must specify the address of the Q.SIG node when your application starts the stack. To do so, specify the address number and type in the ISDN_PROTOCOL_PARMS_Q931CC structure referenced in the **isdnStartProtocol** call. The following fields specify the address:

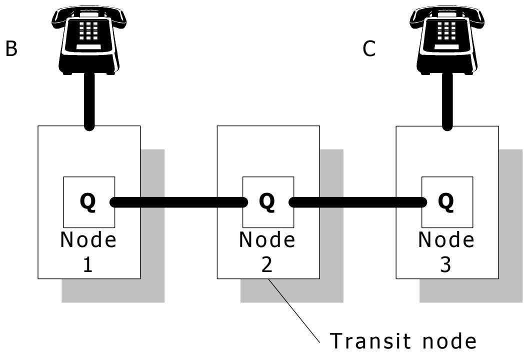
- qsig_source_type_of_nb
- qsig_source_party_nb_type
- qsig_source_addr

For more information about these fields, including a list of possible values, see the *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual*.

5 Tandem supplementary services

Tandem services

Tandem services allow a Q.SIG application to support a transit node. A transit node is a node that exists between two other nodes and routes messages from one node to the other, as shown in the following illustration:

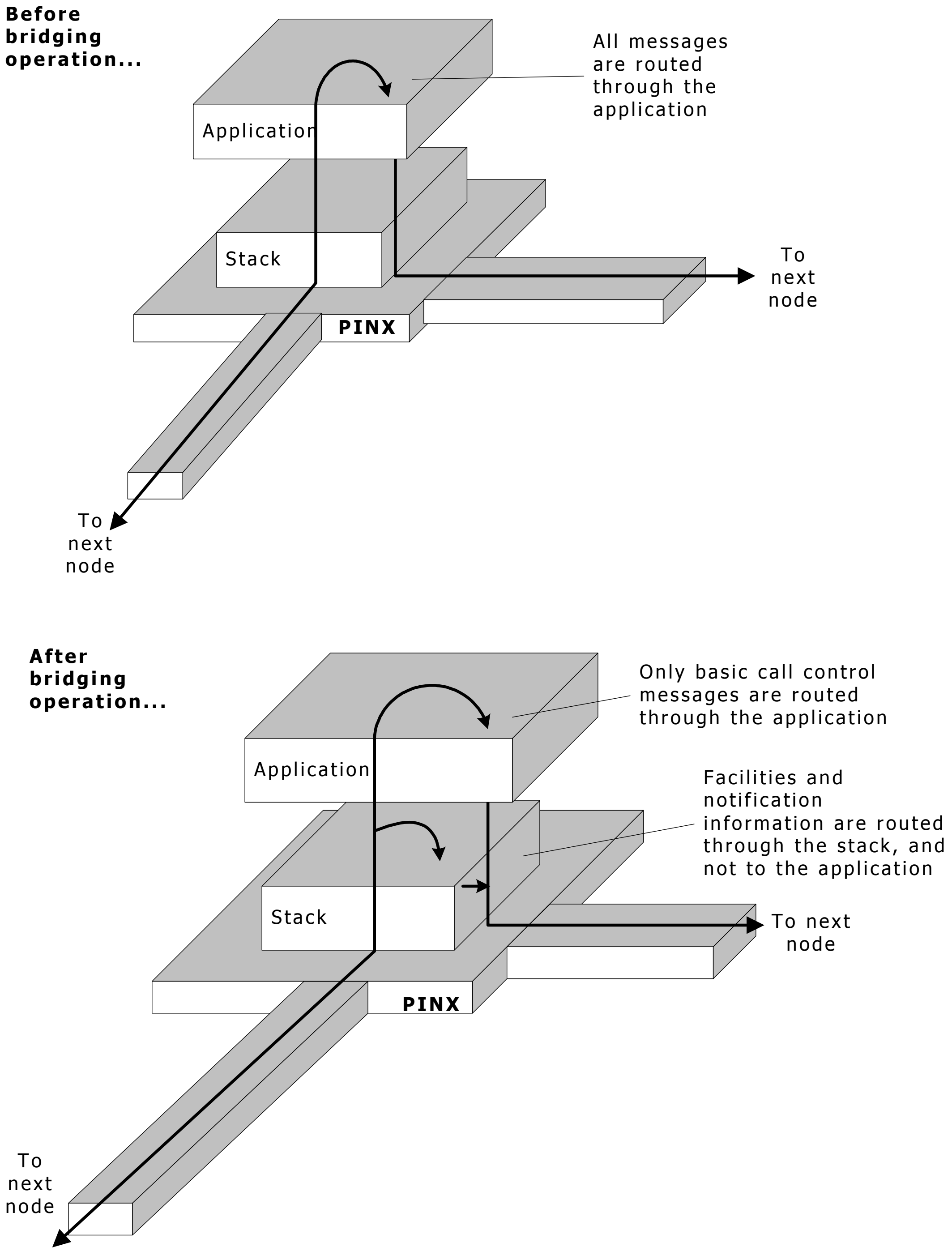


NMS ISDN supports the bridge calls tandem service. This service filters messages routed through the transit node between two served users so the application at the transit node receives only basic call control messages. Facilities and notification information is automatically forwarded.

Bridge calls service

An application invokes the bridge calls service after the local node assumes the role of a transit node between two calls. After this service is invoked, the stack at the local node forwards facilities and notification information between the two legs of the network path, without involving the application. However, basic call control signaling for the two calls (such as call clearing) is still managed by the application on the local node.

The bridge calls service is supported by the Q.SIG variant only.


Before bridging operation...
All messages are routed through the application
Application
Stack
PINX
To next node
To next node
After bridging operation...
Only basic call control messages are routed through the application
Application
Facilities and notification information are routed through the stack, and not to the application
Stack
PINX
To next node
To next node

Call bridging

Invoking bridge calls

To explicitly invoke the bridge calls service, send an acu_ss_bridge_calls_invoke extended data structure to the stack in an ACU request message, invoking the invoke bridge calls operation.

Use the ID of one of the calls as the connection ID. Specify the connection ID of the other call in the bridge_to field in the acu_ss_bridge_calls_invoke supplementary service structure. Only one invocation is required to bridge both connections.

Note: In order for the bridge calls supplementary service to operate properly, the NS_IE_RELAY_BEHAVIOUR bit must be set to its default setting (in the ns_behavior substructure referenced in the call to **isdnStartProtocol**). For more information, see the *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual.*

The bridge calls service can also be invoked implicitly when the application performs a transfer-by-join operation (see *Call transfer and Q.SIG* on page 63).

Successful bridge calls invocation

If the bridge calls invocation is successful, both remote parties receive an acu_ss_bridge_calls_ret_result extended data structure in an ACU confirmation or indication message, indicating that the calls have been bridged.

Once the bridge calls service is invoked, it cannot be disabled. When either of the calls is cleared, the service ends.

Unsuccessful bridge calls invocation

The bridge calls invocation operation is rejected in the following situations:

- One of the connections is not Q.SIG. Bridging between the Q.SIG variant and other variants is not supported.
- The NAIs for the two involved connections are on different boards. Bridging between boards is not supported. In these cases, the application layer must provide the bridging function.
- One or both of the calls is already bridged.
- The connection ID specified in the bridge_to field is not found.

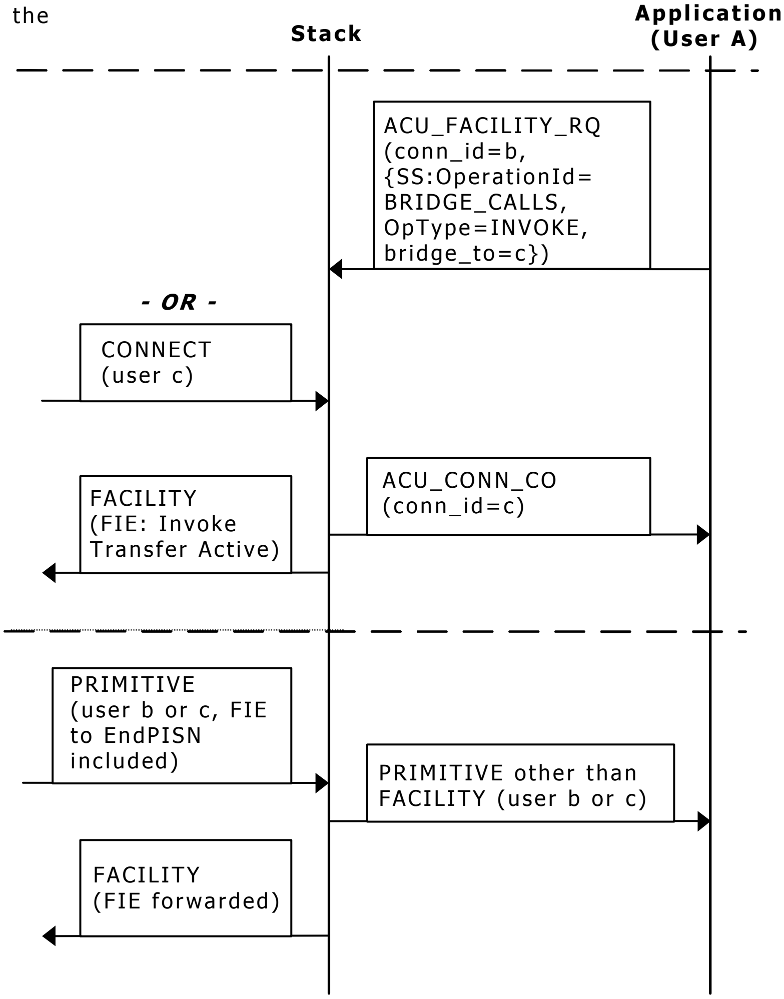
If the bridge calls invocation operation is rejected, an acu_ss_reject extended data structure is sent to the application.

From the perspective of the application on the local node:

The application explicitly invokes call bridging between users B and C.

Alternatively, call bridging is implicitly invoked when the notify transfer service is invoked. In the example at right, A has invoked notify transfer on user C.

Once call bridging is invoked, only basic call control messages (such as ACU_CONN_CO) are sent to the application. If a call control message contains facilities information, this is passed on.


Stack
Application (User A)
ACU_FACILITY_RQ (conn_id=b, {SS:OperationId= BRIDGE_CALLS, OpType=INVOKE, bridge_to=c})
- OR -
CONNECT (user c)
ACU_CONN_CO (conn_id=c)
FACILITY (FIE: Invoke Transfer Active)
PRIMITIVE (user b or c, FIE to EndPISN included)
PRIMITIVE other than FACILITY (user b or c)
FACILITY (FIE forwarded)

6 Notify hold and retrieve services

Notify hold

The notify hold operation notifies a remote user of a call hold operation involving the user. Notify hold is available in an ETS 300 configuration and in a Q.SIG configuration.

Notify hold (ETS 300)

In an ETS 300 configuration, the stack performs this operation to notify the application when the remote equipment has changed the auxiliary state of a local subscriber to the on hold state. An acu_ss_notify_hold_invoke extended data structure is sent to the application in an ACU indication message.

Notify hold (Q.SIG)

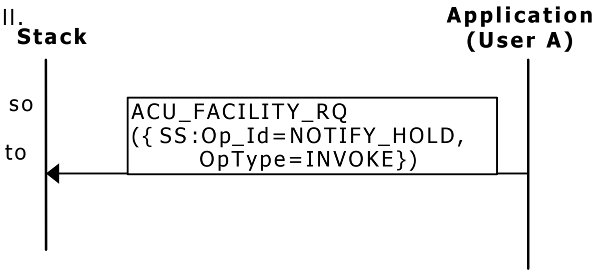
In a Q.SIG configuration, the application at one node performs this operation to notify the application on a remote node when a call from the remote node is on hold. To do so, it sends an acu_ss_notify_hold_invoke extended data structure to the location in an ACU request message.

The following illustrations show the message types interchanged between the stack and a Q.SIG application when notify hold is invoked from the perspectives of the local node and the remote node:

Local notify hold

(From perspective of user A) A and B are in active call.

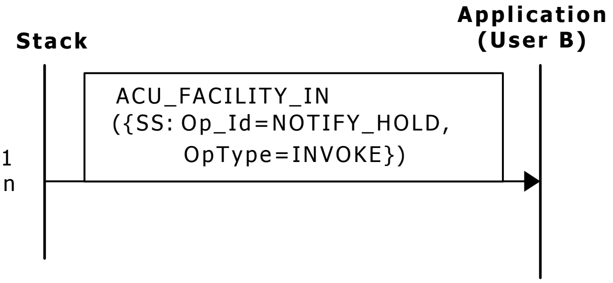
User A invokes hold at PINX 1, so PINX 1 sends a notify message to user B.



Remote notify hold

(From perspective of user B) A and B are in an active call.

User A invokes local call hold on PINX 1. PINX 3 receives message from PINX 1 that says B is now on hold.



Notify retrieve

The notify retrieve operation notifies a remote user of a call retrieve operation involving the user.

Notify retrieve (ETS 300)

In an ETS 300 configuration, the stack performs this operation to notify the application when the remote equipment has changed the auxiliary state of a local subscriber to retrieved. An acu_ss_notify_retrieve_invoke extended data structure is sent to the application in an ACU indication message.

Notify retrieve (Q.SIG)

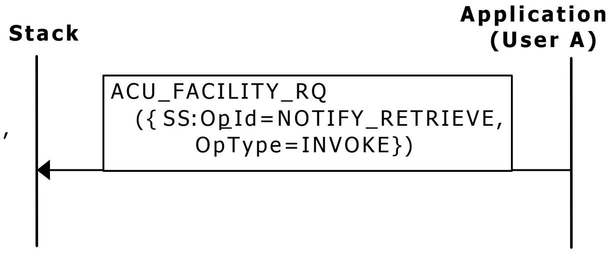
In a Q.SIG configuration, the application at one node performs this operation to notify the application on a remote node when a call previously on hold has been retrieved. To do so, it sends an acu_ss_notify_retrieve_invoke extended data structure to the application in an ACU request message.

The following illustrations show the message types interchanged between the stack and a Q.SIG application when the notify retrieve service is invoked from the perspectives of the local node and the remote node:

Local notify retrieve

(From perspective of user A) A and B are on hold.

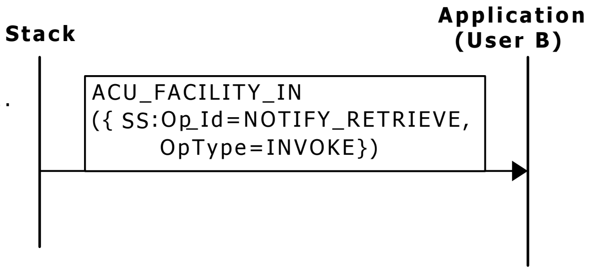
User A invokes retrieval at PINX 1, so PINX 1 sends a notify message to user B.



Remote notify retrieve

(From perspective of user B) A and B are on hold.

User A invokes local call retrieval on PINX 1. PINX 2 receives a message from PINX 1 which says B is now retrieved.



7 Call transfer services

Transfer interface overview

The ISDN transfer interface provides a common, consistent, non-restrictive API for ISDN transfer. The transfer interface does not define fixed behavior for transfer functions, but instead provides a common framework that should be used to implement the transfer service for a given protocol variant.

The ISDN transfer interface includes one API for use with NMS ISDN Messaging and one for use with NMS Natural Call Control.

NMS ISDN Messaging transfer interface

The NMS ISDN Messaging (ACU) transfer interface creates three messages:

- ACU_TRANSFER_RQ
- ACU_CALLID_IN
- ACU_TRANSFER_CO

None of these messages have fixed mapping into Q.931 messages. Their exact mapping depends on the protocol variant being used.

Performing a call transfer

The ACU call transfer messages are used in the following sequence during a call transfer:

Step	Action
1	An outbound call is placed and the callid_rq field is accessed through the Acu_conn_rq_callid_rq macro. **Note:** See the *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual* for more information on the Acu_conn_rq_callid_rq macro.
2	An ACU_CALLID_IN message is received. This message is associated with the outbound call and contains the callid for the second call.
3	An ACU_TRANSFER_RQ message transfers the calls. This message contains the callid for the second call. The transfer request is made on the first call, not on the outbound call that was placed in step 1.
4	An ACU_TRANSFER_CO message is received, reporting the status of the transfer request.

ACU call transfer messages

All three messages use the same data structure for their parameters. The following section of code shows this data structure:

```
#define ACU_MAX_CALLID_SIZE 8

#define TRERR_UNKNOWN      0x01
#define TRERR_NOT_ALLOWED  0x02
#define TRERR_BAD_CALLID   0x03
#define TRERR_FAILED       0x04

struct acu_transfer_args
{
    uchar status;
    uchar callid_present;
    uchar callid[ACU_MAX_CALLID_SIZE];
    pad6
};

#define Acu_transfer_ ((struct acu_transfer_args FAR *)p_data) ->
#define Acu_transfer_status Acu_transfer_ status
#define Acu_transfer_callid_present Acu_transfer_ callid_present
#define Acu_transfer_a_callid Acu_transfer_ callid

#define Acu_transfer_size     (Rnd_sizeof(struct acu_transfer_args))
```

ACU_TRANSFER_RQ message

The ACU_TRANSFER_RQ message is an application's request for transfer. The following table provides a description of the ACU_TRANSFER_RQ associated fields:

Field	Description
status	This field is ignored for this message.
callid	The identity of the second call involved in the transfer request.
callid_present	A value of 1 indicates that the callid field contains a valid value.

ACU_CALLID_IN message

The ACU_CALLID_IN message returns the callid produced by the stack, or reports a failure. The following table provides a description of the ACU_CALLID_IN associated fields:

Field	Description
status	Contains 0 if delivery succeeds and an error code upon failure.
callid	The identity of the second call involved in the transfer request.
callid_present	A value of 1 indicates that the callid field contains a valid value.

ACU_TRANSFER_CO message

The ACU_TRANSFER_CO message returns the status of the requested transfer. The following table provides a description of the ACU_TRANSFER_CO associated fields:

Field	Description
status	Contains 0 if transfer succeeds and an error code upon failure.
callid	For NI2/TBCT this will contain the call tag for the notification to controller (NTC) feature. For other protocols this field is unused. **Note:** The call tag is not the callid.
callid_present	A value of 1 indicates that the callid field contains a valid value.

ACU call transfer error codes

If either callid delivery or call transfer fails, the status field of the message contains one of the following error codes:

Status error name	Value	Meaning
TRERR_UNKNOWN	0x01	Reason for failure is unknown.
TRERR_NOT_ALLOWED	0x02	Transfer or callid request is not allowed.
TRERR_BAD_CALLID	0x03	Callid field contains an invalid value.
TRERR_FAILED	0x04	General failure.

The callid_rq field in the acu_conn_rq_args structure enables a request for a callid when a call is placed. If the callid_rq field is set to 0, no request for a callid is made. Otherwise, the request is sent.

The callid_rq field is accessed through the Acu_conn_rq_callid_rq macro, as described in the *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual*.

Natural Call Control (NCC) transfer interface

To transfer a call using the Natural Call Control transfer interface, one of the calls must have a callid. This topic describes how to obtain a callid and how to initiate a call transfer using the NCC transfer interface.

Obtaining a callid

There are two ways to obtain a callid:

- Set the getcallid field in the PLACECALL_EXT structure to 1, enabling the request of a callid upon outbound call setup.

```
typedef struct
{
  DWORD size;                           /* size of this structure           */
 . . .
  DWORD getcallid;                      /* get callid for this call         */
 . . .
} PLACECALL_EXT;
```

- The application invokes **nccSendCallMessage** for one of the existing calls to be transferred. In the function invocation request, the message argument should point to an NCC_ISDN_SEND_CALL_MESSAGE structure. The size argument should be set to the size of the structure in which message_id = NCC_ISDN_TRANSFER_CALLID_RQ and message_type = 0.

  This method is required to obtain a callid for an inbound call.

```
typedef struct
{
  WORD message_id;
  WORD message_type;
} NCC_ISDN_SEND_CALL_MESSAGE;
```

Note: Some protocol variants do not support this method of obtaining a callid.

In either case, the application is informed (by the NCCEVN_CALLID_AVAILABLE event) when a callid is available. Any time after this occurs, the application can initiate a call transfer.

Initiating call transfer

There are two ways to initiate call transfer:

- The application invokes **nccTransferCall** with the call handles of the two calls to be transferred. Using this method, the application has only to request a callid. The callid is managed and used by NCC.
- An application can be written with several contexts and processes (or several machines) and can be required to transfer two calls that are each managed by different processes. In this situation, each process has the call handle for one of the calls, but not the other. Neither process can call **nccTransferCall** with both call handles.
  In this situation, **nccTransferCall** can be invoked with one call handle and a second call handle of 0, as long as the TRANSFERCALL_EXT structure is supplied. For example:

```
typedef struct
{
DWORD size;                      /* size of this structure */
WORD id[NCC_ISDN_CALLID_LEN];    /* call identifier        */
} TRANSFERCALL_EXT;
```

Set the size field to the size of the structure and the id field to the callid for the other leg of the transfer. The callid is available to the other process in the callid field of the NCC_ISDN_EXT_CALL_STATUS structure after the NCCEVN_CALLID_AVAILABLE event is received. For example:

```
typedef struct
{
. . .
WORD callid[NCC_ISDN_CALLID_LEN];  /* call identifier          */
. . .

} NCC_ISDN_EXT_CALL_STATUS;
```

Note: For NCC, transfer can be performed only on calls on two separate B channels.

When a transfer is successful, the application receives the NCCEVN_CALL_DISCONNECTED message, with the reason code NCC_DIS_TRANSFER for the call that was transferred.

If call transfer fails, the application receives the NCCEVN_PROTOCOL_ERROR message with a value of NCC_PROTERR_TCT_FAILED and the size field is set to one of the error codes given in NMS ISDN messaging transfer interface. For more information, refer to the *Dialogic® NaturalAccess™ ISDN Software Developer's Manual*.

Call transfer and ETS 300

ETS 300 user-side specifications describe interactions between a user (an application) and the network from the user's point of view. In an ETS 300 configuration, the network performs the actual transfer operation (connects the two remote parties together). At least one of the parties in the transfer must have invoked the hold service.

The application's role in a call transfer is to request the network to perform the transfer using the explicit call transfer service. As shown in the following illustrations, once the transfer is complete, affected connection IDs and B channels over the S/T reference point are cleared so the local application is no longer involved in the call.

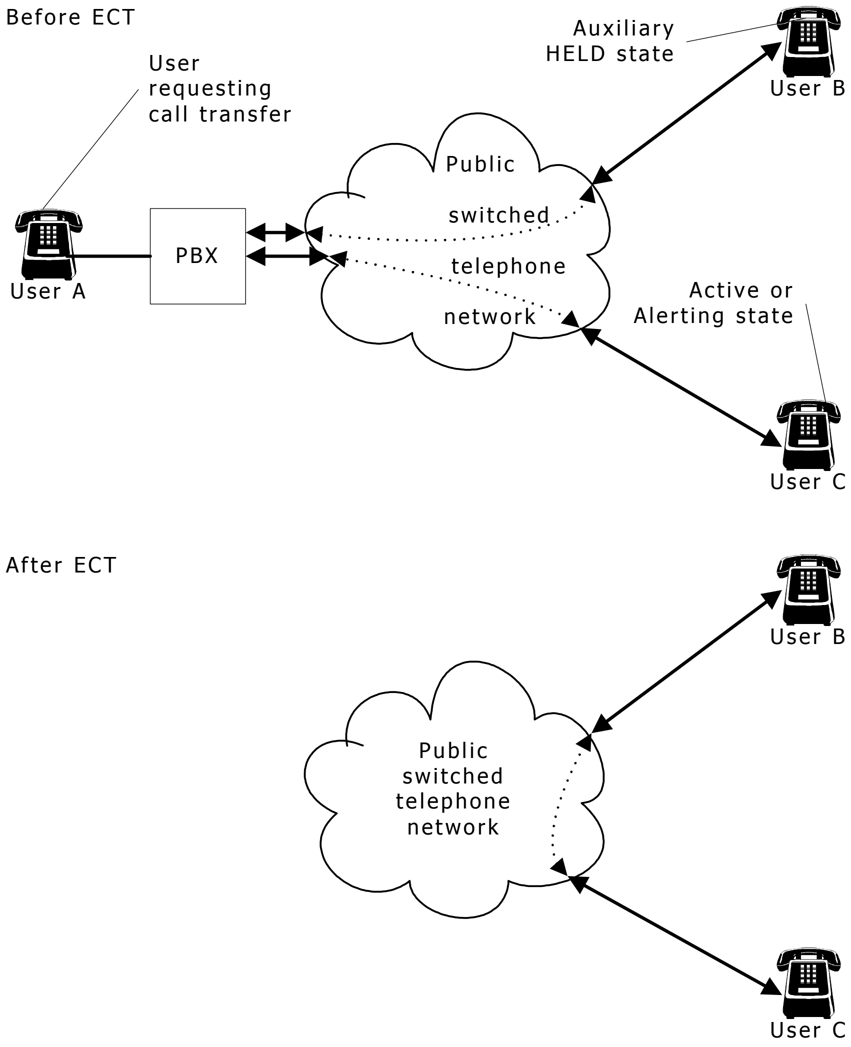
Note: After the ECT service is invoked, the user side cannot recover the call from the network.

When an explicit call transfer succeeds, the network informs the remote parties of the transfer using a notify transfer operation.

The application is responsible for assuring that the joined parties are compatible. For example, both parties should be carrying the same type of service (data or voice).

Performing an explicit call transfer (ETS 300 only)

To perform a call transfer, an ETS 300 application invokes the explicit call transfer (ECT) service. This service allows an application serving two calls, one of which is in the auxiliary HELD state, to request the network to join the two remote calls together and drop the local connection.



Before ECT

After ECT

User requesting call transfer
Auxiliary HELD state
User B
Public switched telephone network
PBX
User A
Active or Alerting state
User C
User B
Public switched telephone network
User C

Call transfer and Q.SIG

Q.SIG specifications describe the behavior of nodes in a network and how they interact. NMS ISDN supports two types of call transfer operations between Q.SIG nodes:

- Transfer-by-rerouteing
- Transfer-by-join

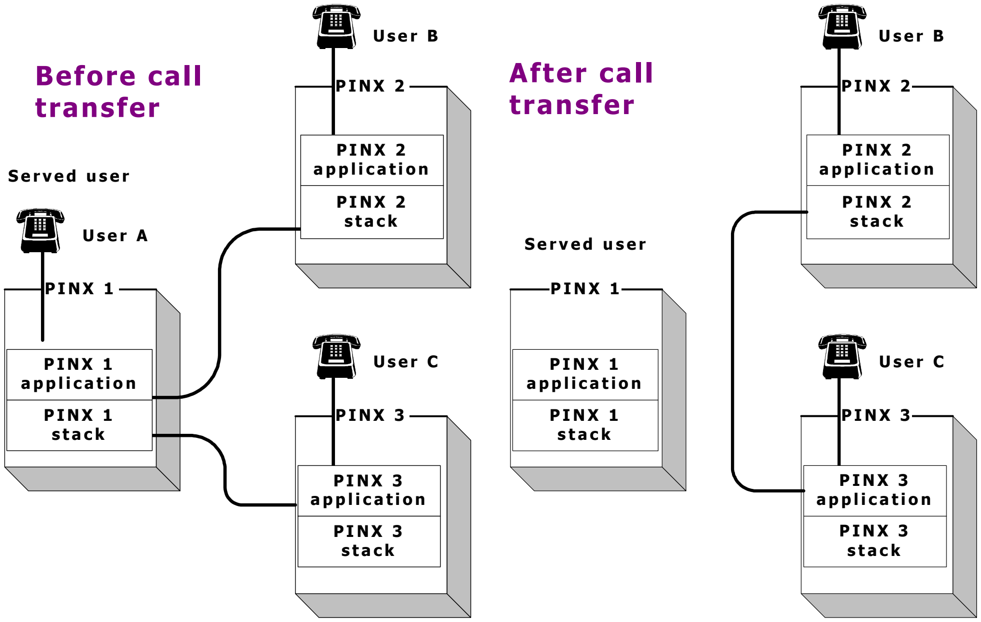
Both operations are invoked using either **nccTransferCall** in the Natural Call Control service, or the ACU_TRANSFER_RQ message in the ISDN API. To invoke the transfer operation, the application must do the following:

Step	Action	Result
1	Obtain the call ID for transfer.	The stack attempts to get the call ID for transfer by using the transfer-by-rerouteing operation. If this attempt fails, the stack uses transfer-by-join.
2	Request the transfer.	The stack transfers the call by using the same operation it used to obtain the caller ID. For example, if the stack used the transfer-by-join operation to obtain the caller ID, it also uses this method for the transfer itself.

Transfer-by-rerouteing

In a transfer-by-rerouteing operation, the two separate calls are connected outside the local node. After the transfer confirmation, the local node is no longer involved with the transferred calls.

The following illustration shows how the transfer-by-rerouteing operation works:



Transfer-by-join

In a transfer-by-join operation, the two separate calls are connected through the local node, and the local node remains involved in the call (for example, the call is not rerouted).

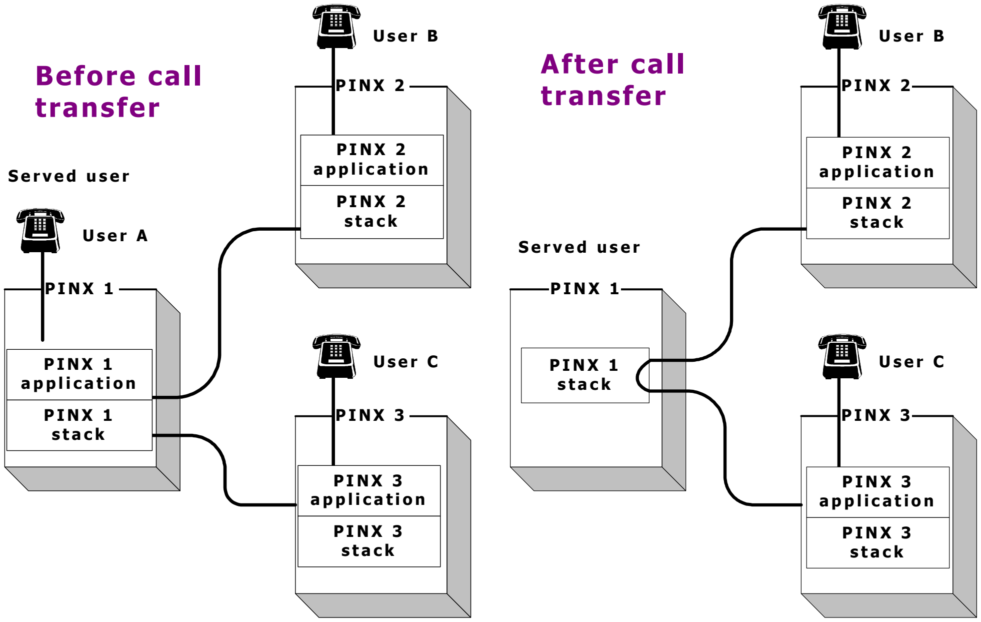
To transfer a call, the application must perform the switching required to connect calls together by using standard calls to the Natural Access Switching service. Then, the application must invoke the transfer operation.

Once the transfer is invoked, the stack at the local node allows the two separate calls connected through the node to pass facilities information to each other without involving the local application. Basic call control signaling for the two calls (such as call clearing) must still be managed by the application. In effect, the calls are bridged (see *Tandem services* on page 27).

The application is responsible for assuring that the B and C parties that are joined are compatible. For example, both parties must be carrying the same type of service (data, voice).

Note: For the call bridging to be successful, the NS_IE_RELAY_BEHAVIOUR bit must be set to its default setting in the ns_behavior substructure referenced in the call to **isdnStartProtocol**. For more information, see the *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual*.

The following illustration shows how the transfer-by-join operation works:



Call transfer and NI2

NI2 user-side specifications describe interactions between an application and the network from a user's point of view. In an NI2 configuration, the network performs actual transfer operations. The application's role in a call transfer is to send a request to the network to perform TBCT using the enhanced explicit call transfer supplementary service. This topic describes:

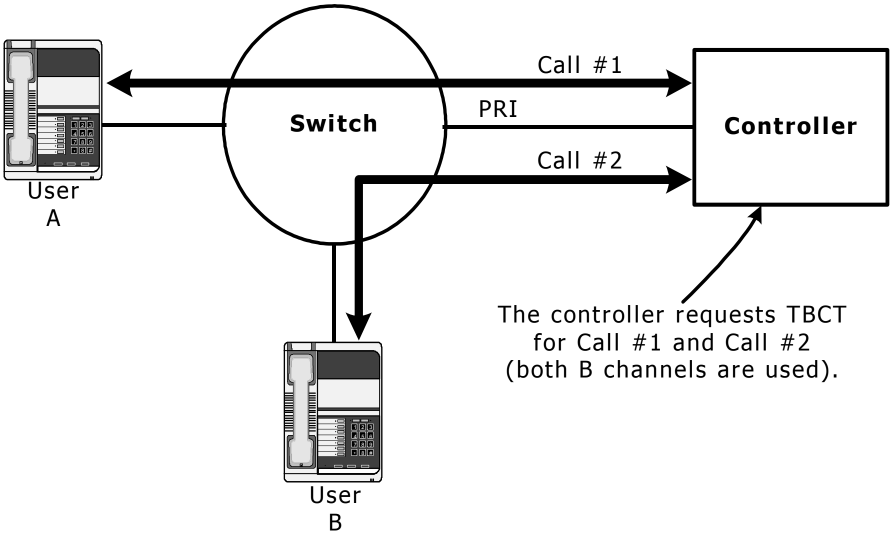
- Invoking two B channel transfer (NI2)
- TBCT notification (NI2) and the ACU_NOTIFY_IN message

The following terms are used in the illustrations in this topic:

This term...	Refers to...
Call #1	The call between the controller and user A.
Call #2	The call between the controller and user B.
Controller	The application requesting TBCT.
User A	The first user connected with the controller that is to be transferred to user B. User A does not need to be an ISDN terminal or connected to an ISDN exchange.
User B	The second user connected with the controller that the transferred user (user A) is to be connected to upon completion of TBCT. User B does not need to be an ISDN terminal or connected to an ISDN exchange.

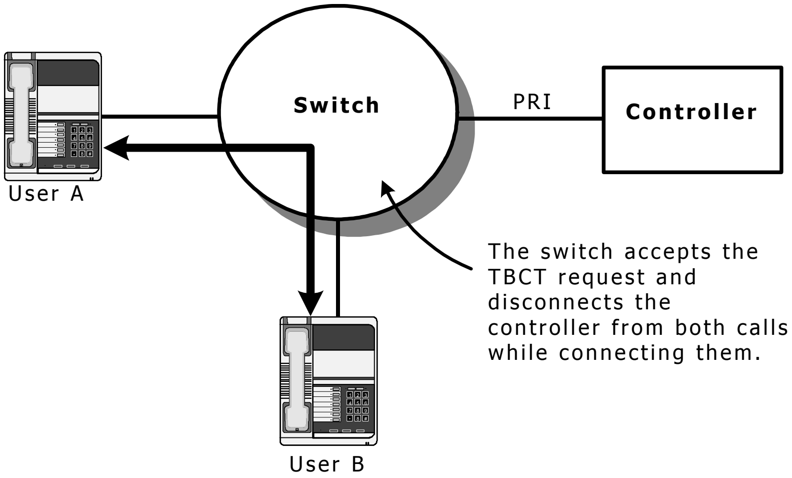
A controller requests TBCT for two calls

TBCT enables a controller on a PRI network to request that the switch connect two independent calls. The two calls can be served by the same PRI trunk or by two different PRI trunks that both serve the application, as shown in the following illustration:


Call #1
PRI
Call #2
Switch
Controller
User A
User B
The controller requests TBCT for Call #1 and Call #2 (both B channels are used).

The switch accepts and processes the TBCT request

If the switch accepts the request, the controller is released from the calls and the other two users are directly connected, as shown in the following illustration:


Switch
PRI
Controller
User A
User B
The switch accepts the TBCT request and disconnects the controller from both calls while connecting them.

TBCT works only when all of the following conditions are met:

- The controller has subscribed to TBCT.
- The controller has at least two independent calls.
- At least one call to be transferred has been answered.

If the other call is...	Then the call...
Outgoing from the controller	Is alerting or has been answered.
Incoming to the controller	Has been answered.

- The bearer capabilities of both calls are compatible.

Note: Your application is responsible for assuring that the bearer capabilities of both calls are compatible.

The switch notifies a controller when a transferred call is cleared if the controller subscribes to the notification to controller feature.

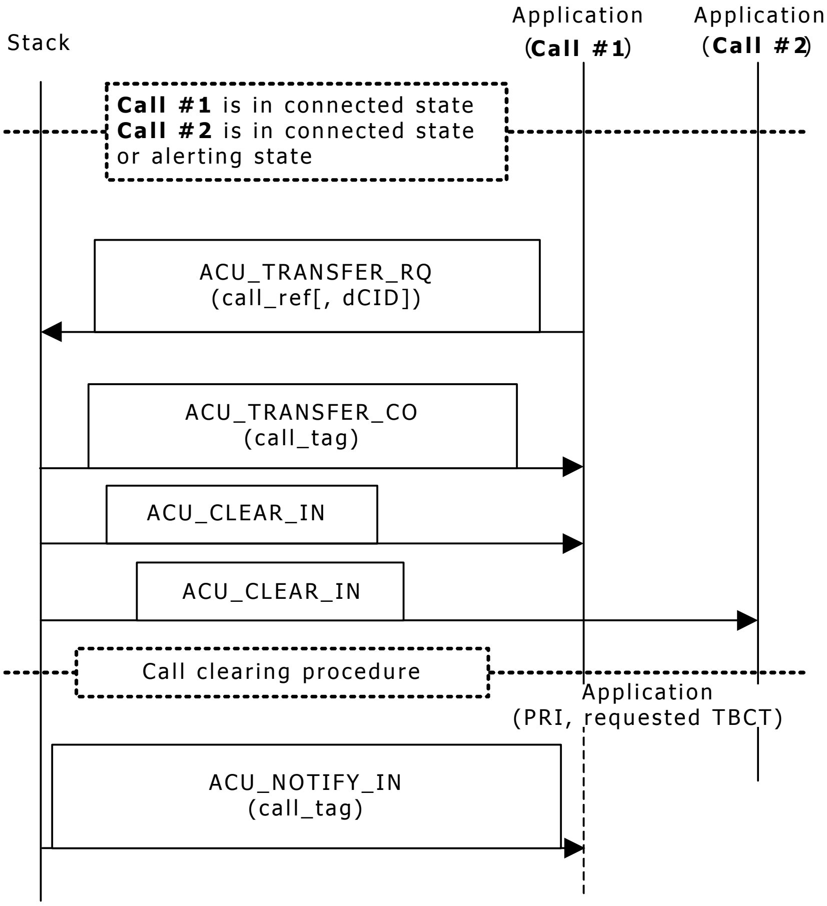
Invoking two B channel transfer (NI2)

To perform a two B channel transfer, an NI2 application invokes the enhanced explicit call transfer supplementary service. This service allows an application serving two calls, one in the connected state and other in either the connected state or the alerting state, to request the network to connect the remote parties of these two calls and drop the local connections.

TBCT notification (NI2)

If the PRI serving the controller subscribed to receive the notification to controller (NTC) feature, the controller receives a call tag for each of the calls corresponding to the transferred parties when these calls are cleared from the PRI. The call tag is associated with the transferred call until it is cleared and is unique to that call for that PRI.

Note: If the controller is served by more than one PRI, the same call tag can be used for different transferred calls on different PRIs.


Stack
Application (Call #1)
Application (Call #2)
Call #1 is in connected state
Call #2 is in connected state or alerting state
ACU_TRANSFER_RQ (call_ref[, dCID])
ACU_TRANSFER_CO (call_tag)
ACU_CLEAR_IN
ACU_CLEAR_IN
Call clearing procedure
Application (PRI, requested TBCT)
ACU_NOTIFY_IN (call_tag)

ACU_NOTIFY_IN message

The ACU_NOTIFY_IN message contains an acu_ss_notify_tbct_calls_ret_result extended data structure. This message means that the transferred call was cleared and the controller should release the call tag value for re-use.

Call transfer and DMS

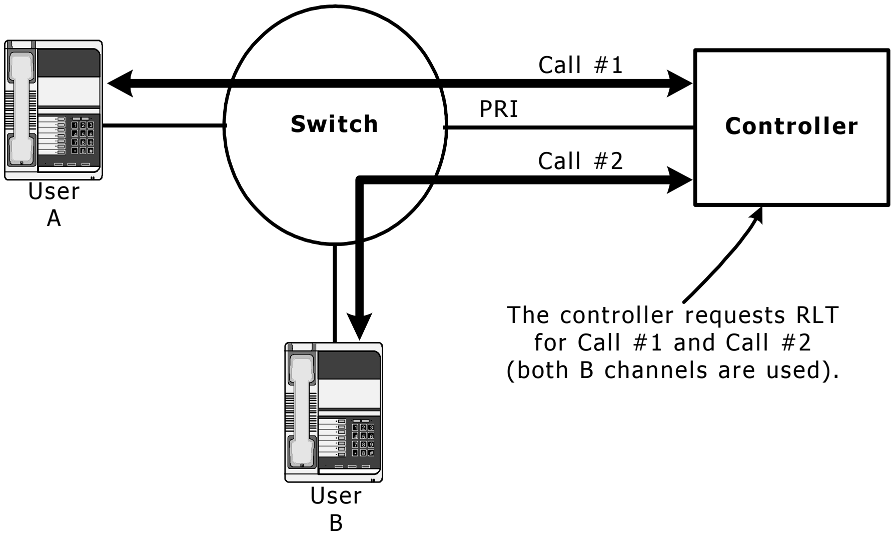
DMS user-side specifications describe interactions between an application and the network from a user's point of view. In a DMS configuration, the network performs actual transfer operations. The application's role in a call transfer is to send a request to the network to perform a release link trunk (RLT) operation.

The following terms are used in the illustrations in this topic:

This term...	Refers to...
Call #1	The call between the controller and user A.
Call #2	The call between the controller and user B.
Controller	The application requesting RLT, using the user-side DMS protocol.
User A	The first user connected with the controller that is to be transferred to user B. User A does not need to be an ISDN terminal or connected to an ISDN exchange.
User B	The second user connected with the controller that the transferred user (user A) is to be connected to upon completion of RLT. User B does not need to be an ISDN terminal or connected to an ISDN exchange.

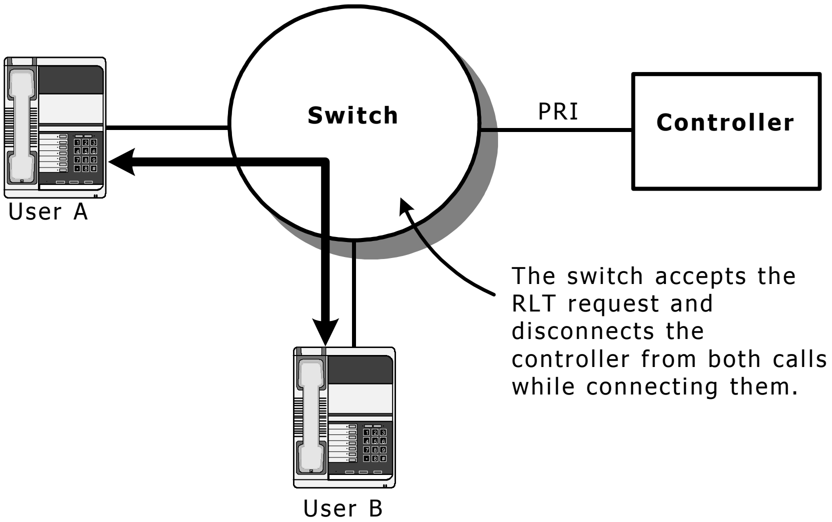
A controller requests RLT for two calls

RLT enables a controller on a PRI to request that the switch connect two independent calls. The two calls can be served by the same PRI trunk or by two different PRI trunks that both serve the application, as shown in the following illustration:


Call #1
PRI
Switch
Controller
Call #2
User A
User B
The controller requests RLT for Call #1 and Call #2 (both B channels are used).

The switch accepts and processes the RLT request

If the switch accepts the request, the controller is released from the calls and the other two users are directly connected, as shown in the following illustration:


Switch
PRI
Controller
User A
User B
The switch accepts the RLT request and disconnects the controller from both calls while connecting them.

RLT works only when all of the following conditions are met:

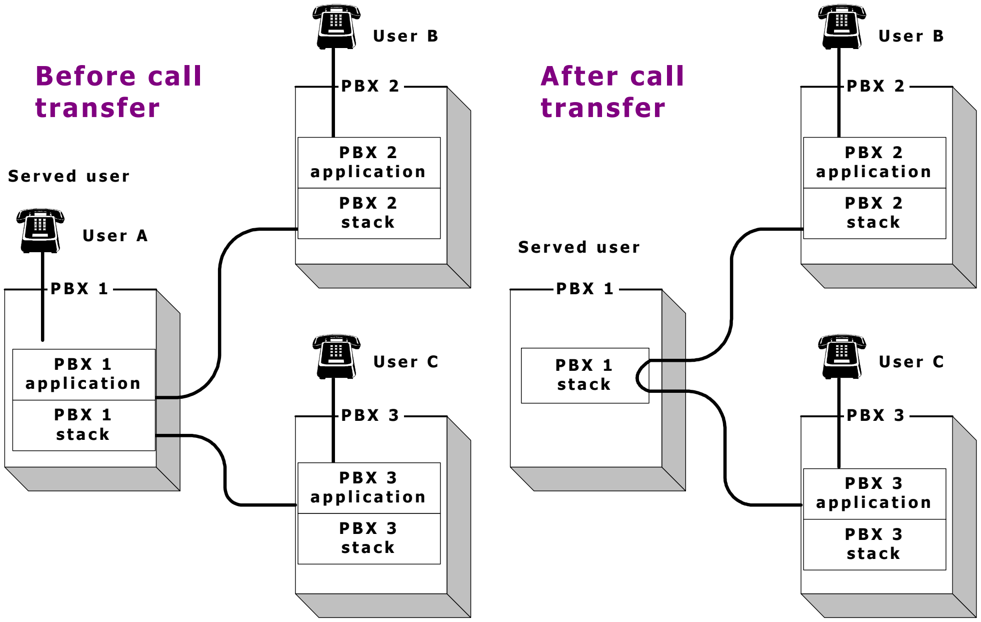
- The controller has subscribed to RLT.
- The controller has at least two independent calls.
- The bearer capabilities of both calls are compatible.
- Both calls are in the connected state at the time RLT is invoked.
- Call #2 is an outbound call originated by the controller. The call origination must specify that the call may become a candidate for RLT, by setting the getcallid field in the PLACECALL_EXT structure.
- Call #1 can be either inbound or outbound.
- Call #1 can be established either before or after call #2.

Call transfer and DPNSS

DPNSS specifications describe the behavior of nodes in a network and how they interact. NMS ISDN supports transfer-by-join operations between DPNSS nodes. In a transfer-by-join operation, the two separate calls are connected through the local node, and the local node remains involved in the call (for example, the call is not rerouted).

To transfer a call under DPNSS, the application must perform the switching required to connect calls together using standard calls to the Natural Access Switching service. The application also invokes a notify transfer operation to notify each remote party that the transfer took place.

If both parties are remote, once the notification takes place, the stack at the local node allows the two separate calls connected through the node to pass facilities information to each other without involving the local application. Basic call control signaling for the two calls (such as call clearing) must still be managed by the application. In effect, the calls have been bridged.



The application is responsible for assuring that the B and C parties that are joined are compatible. For example, both parties should be carrying the same type of service (data, voice).

The two channels to be transferred must be on the same board.

8 Call forwarding services

Call forwarding services overview

This section describes supplementary services related to call forwarding. Call forwarding is a service performed at a served user (or on behalf of a served user) to reroute a call to another user. This topic describes:

- Call forwarding in an ETS 300 application
- Call forwarding in a Q.SIG application

The following services are related to call forwarding:

Service	Description
Call diversion	Calls addressed to a served user are rerouted (forwarded) to a different address by the network, on behalf of the served user. Available under ETS 300 and Q.SIG. Call diversion services can work in any of three ways: Call forwarding - no response (CFNR). Calls can be rerouted to a different user if the originally addressed user fails to answer the call within a certain interval. Call forwarding - busy (CFB). Calls can be rerouted to a different user if the originally addressed user is currently in a call (for example, the line is busy). Call forwarding - unconditional (CFU). Calls can be rerouted to a different user unconditionally.
Call deflection	Allows the served user's stack or application to forward a call to a different address. Available under ETS 300 only.
Remind diversion	When a served user initiates an outbound call, this service informs the served user if call diversion services are activated for incoming calls. Available under ETS 300 only.

Call forwarding in an ETS 300 application

In an ETS 300 application, call forwarding can be accomplished in any of three ways, depending upon which entity the served user performs the forwarding operation (see the following illustrations):

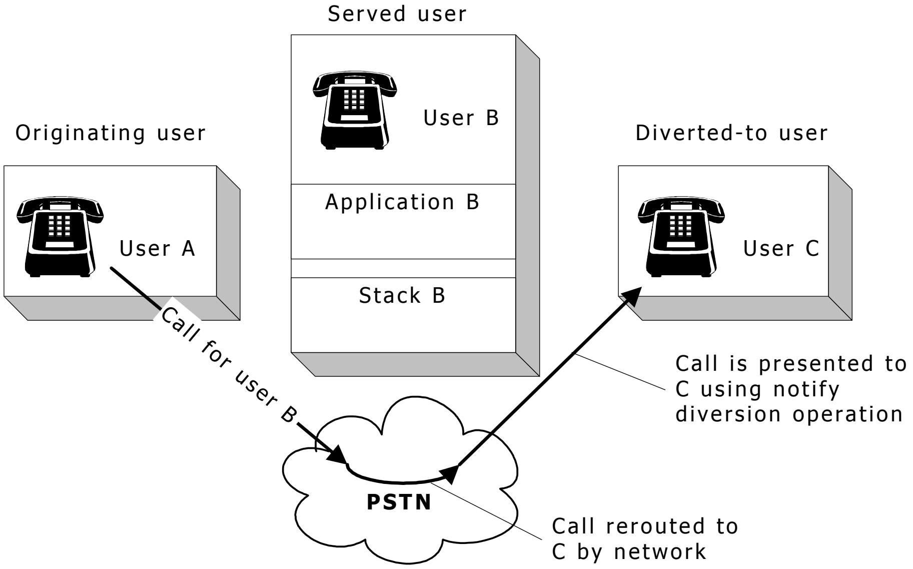
- The served user application can activate or deactivate the call diversion service for a particular user address. Call diversion takes place at the network level. From then until the service is deactivated, if a call arrives for the user, the network reroutes the call to another user.
- The served user application can activate or deactivate the call deflection service for one or more NAIs. When activated, call deflection takes place in the stack. From then until the service is deactivated, all calls on those NAIs are rerouted to another user.
- The served user application can invoke the call deflection service on a call-by-call basis. When the service is invoked in this way, temporary activation of the service occurs implicitly.

Regardless of which service is used, the diverted-to user is always notified of a forwarded call using a notify diversion operation.

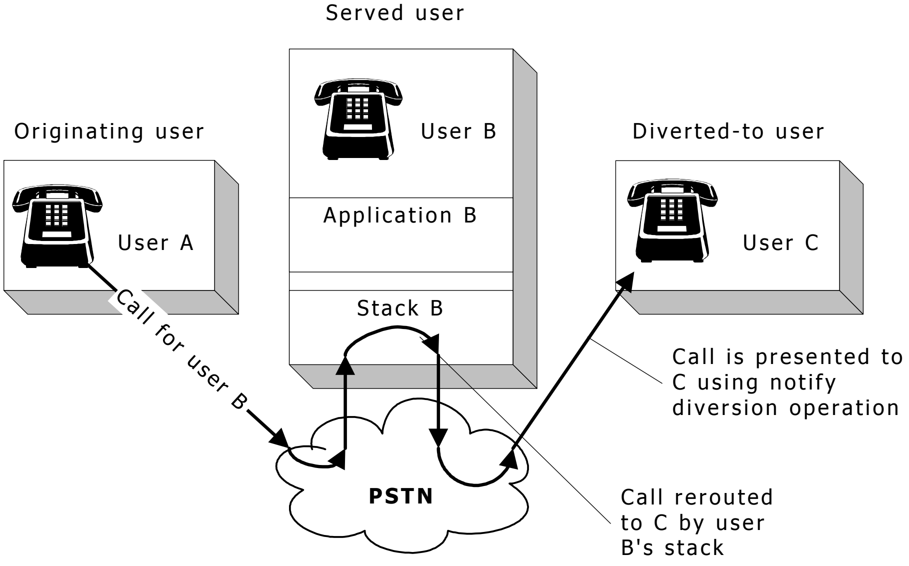
The served user application can enquire the network for the status of the call diversion service for a given user, or for all users.

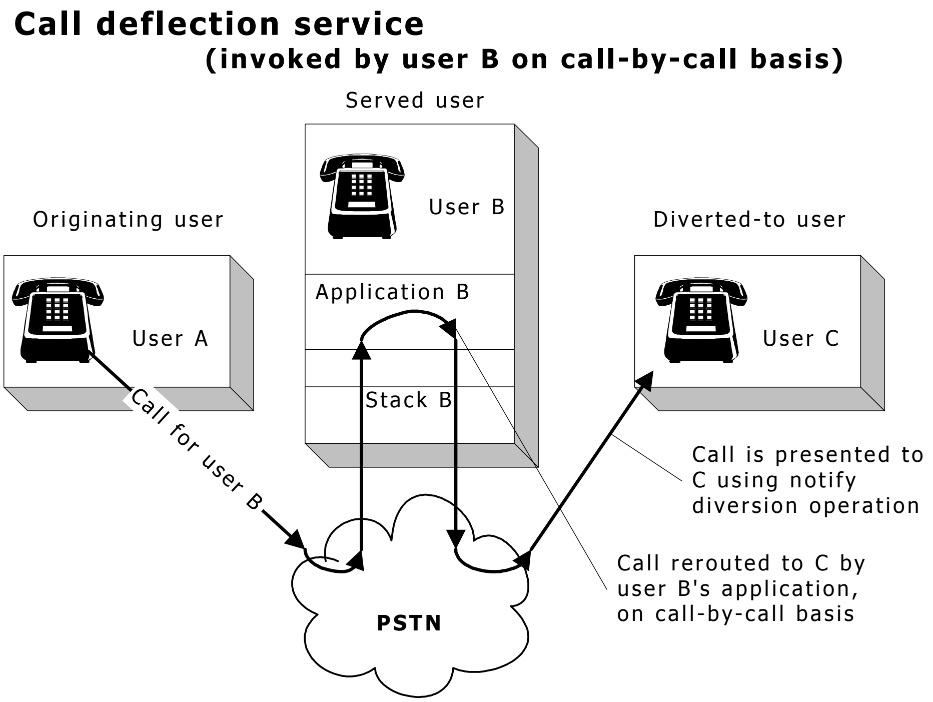
The remind diversion service can be invoked by the network if the user has subscribed to this service. When a served user initiates an outbound call, this service informs the served user if call diversion services are active for incoming calls.

Call diversion service (activated by user B)



Call deflection service (activated by user B)




Call deflection service
(invoked by user B on call-by-call basis)
Served user
User B
Application B
Stack B
Originating user
User A
Call for user B
Diverted-to user
User C
Call is presented to C using notify diversion operation
Call rerouted to C by user B's application, on call-by-call basis
PSTN

Call forwarding in a Q.SIG application

In a Q.SIG application the call forwarding process is as follows:

- The served user node invokes the call diversion service to request the originating node to forward a call.
- The originating node offers the call to the diverted-to user node, performing a notify diversion operation to notify the node that the call has been forwarded from another user.

The Remind diversion service (ETS 300) describes how an application can perform these procedures. Since the application may be associated with the originating node, served user node, or diverted-to node, all three perspectives are considered.

Note: Q.SIG specifications identify a fourth party involved in a diversion: the node responsible for rerouting the call (the rerouting node). NMS ISDN supports the combination of the originating and rerouting nodes in the same node (for example, call diversion by rerouting). It does not support the combination of rerouting node and served user node (for example, call diversion by forward transfer).

Call deflection supplementary services are not applicable to Q.SIG applications since there is no user side of a Q.SIG link.

9 Call diversion services (ETS 300)

Using call diversion services (ETS 300)

ETS 300 call diversion procedures can be viewed as a remote-control mechanism to manage the diversion services performed in the network. The messages are command/response transactions, where the command can be:

- Activating call diversion
- Deactivating call diversion
- Determining the current state of the call diversion service (enquiring)
- Informing a user that a call diversion has taken place (notification)

The response acknowledges the successful processing of the command, and may contain information requested by the command.

Activating call diversion (ETS 300)

To activate the call diversion service for an address, the application sends an acu_ss_act_divert_invoke extended data structure to the stack in an ACU request message using the ACU_MGMT_SAPI.

Successful call diversion activation request (ETS 300)

If the invocation is successful, the stack responds with an ACU indication message using the ACU_MGMT_SAPI. This message contains an acu_ss_act_divert_ret_result extended data structure.

Unsuccessful call diversion activation request (ETS 300)

If the invocation is not successful, the stack responds with an ACU indication message using the ACU_MGMT_SAPI. This message contains an acu_ss_act_divert_ret_error extended data structure.

Deactivating call diversion (ETS 300)

To deactivate the call diversion service for an address, the application sends an ACU request message to the stack containing an acu_ss_deact_divert_invoke extended data structure, using the ACU_MGMT_SAPI.

Successful call diversion deactivation request (ETS 300)

If the request is successful, the stack responds with an acu_ss_deact_divert_ret_result extended data structure in an ACU indication message using the ACU_MGMT_SAPI.

Unsuccessful call diversion deactivation request (ETS 300)

If the request is unsuccessful, the stack responds with an acu_ss_deact_divert_ret_error extended data structure in an ACU indication message using the ACU_MGMT_SAPI.

Enquire diversion operation (ETS 300)

An application can query the network to obtain the following call diversion service information:

- The current state of the specified CFU, CFNR, or CFB procedure for a specific user

or

- A list of all served users with diversion descriptions.

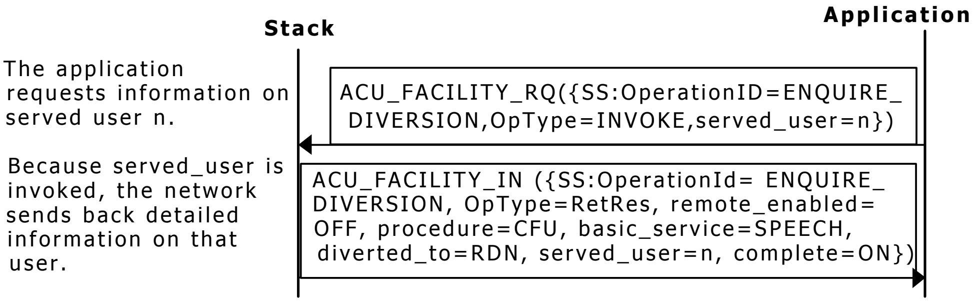
To query the network for status information, the application sends an acu_ss_enquire_divert_invoke extended data structure to the stack in an ACU request message, using the ACU_MGMT_SAPI.

Successful enquire diversion request (ETS 300)

If the enquire diversion operation invocation succeeds, the network responds based on whether or not the served_user field is invoked.

If the served_user field is invoked, the network responds with a single acu_ss_enquire_divert_ret_result extended data structure. Each structure contains detailed information for the served user.

The complete field is set to ON, indicating that all requested information arrived.


Stack
Application
The application requests information on served user n.
ACU_FACILITY_RQ({SS:OperationID=ENQUIRE_DIVERSION,OpType=INVOKE,served_user=n})
Because served_user is invoked, the network sends back detailed information on that user.
ACU_FACILITY_IN ({SS:OperationId= ENQUIRE_DIVERSION, OpType=RetRes, remote_enabled=OFF, procedure=CFU, basic_service=SPEECH, diverted_to=RDN, served_user=n, complete=ON})

Enquire diversion (single served user)

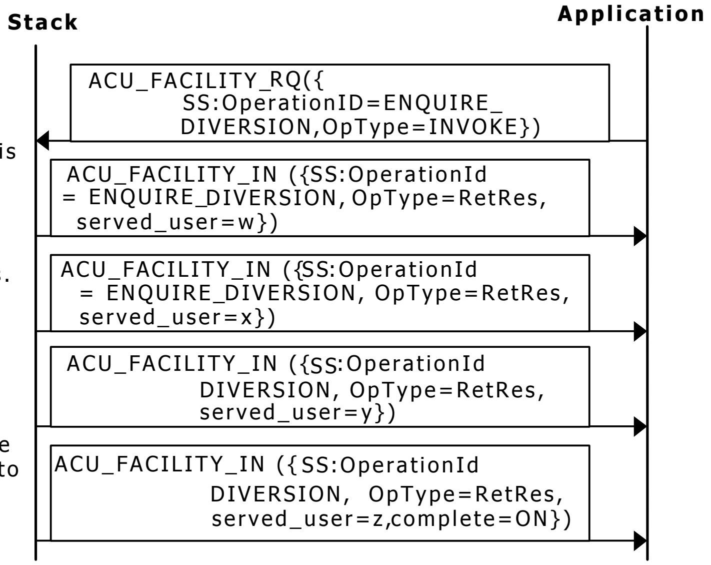
If the served_user field is not invoked, the network responds with one or more acu_ss_enquire_divert_ret_result extended data structures. Each structure contains the address of one served user that has call diversion activated.

In the extended data structure for the last number, the complete field is set to ON. The application can check this field to determine when the information arrives.

The application requests information without specifying a served user.

Because served_user is not invoked, the network sends back multiple responses, each containing one served user's address.

In the last record, the complete field is set to ON.


Stack
Application
ACU_FACILITY_RQ({ SS:OperationID=ENQUIRE_ DIVERSION,OpType=INVOKE})
ACU_FACILITY_IN ({SS:OperationId = ENQUIRE_DIVERSION, OpType=RetRes, served_user=w})
ACU_FACILITY_IN ({SS:OperationId = ENQUIRE_DIVERSION, OpType=RetRes, served_user=x})
ACU_FACILITY_IN ({SS:OperationId DIVERSION, OpType=RetRes, served_user=y})
ACU_FACILITY_IN ({SS:OperationId DIVERSION, OpType=RetRes, served_user=z,complete=ON})

Enquire diversion (all served users)

Unsuccessful enquire diversion request (ETS 300)

If the network encounters an error, an acu_ss_enquire_divert_ret_error extended data structure is returned in an ACU indication message.

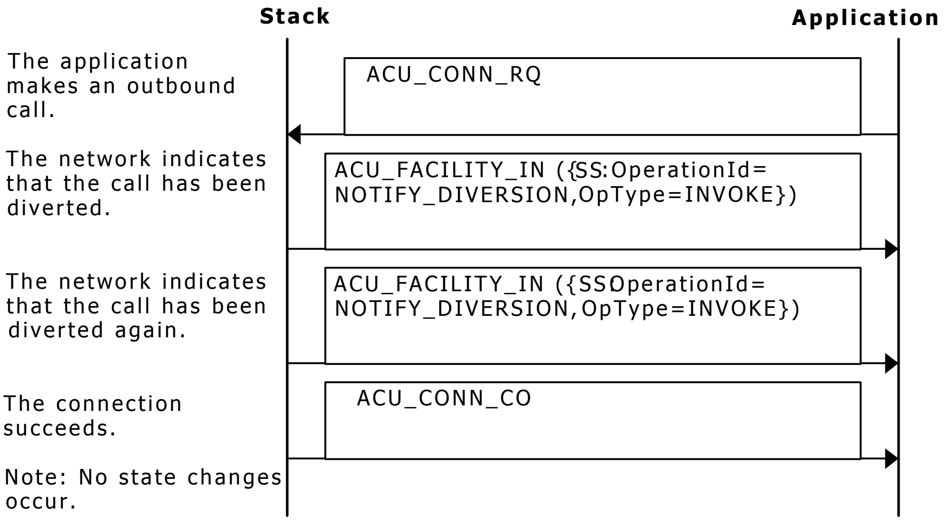
Diversion has taken place (ETS 300)

The stack notifies the application when it learns that an inbound or outbound call has been diverted.

Notification of a diverted outbound call (ETS 300)

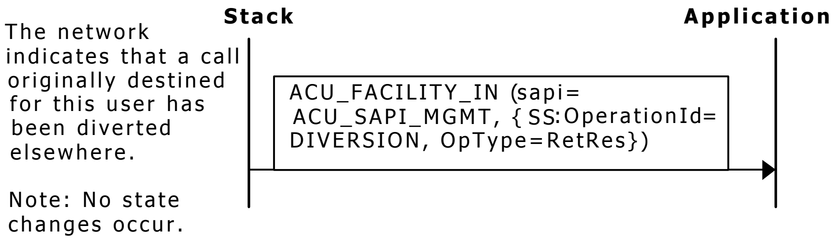
When an outbound call is diverted by the network, the application is notified. An ACU indication message is sent to the application containing an acu_ss_notify_diversion_invoke extended data structure. The connection ID of the outbound call is included in the data structure.

Multiple diversion operations may cause several of these primitives to be sent to the application, as additional information is provided by the network (see the following illustration). If the offered call is diverted multiple times, the fields in the primitive may contain new information that should supersede previous information.


Stack
Application
The application makes an outbound call.
ACU_CONN_RQ
The network indicates that the call has been diverted.
ACU_FACILITY_IN ({SS:OperationId= NOTIFY_DIVERSION,OpType=INVOKE})
The network indicates that the call has been diverted again.
ACU_FACILITY_IN ({SSOperationId= NOTIFY_DIVERSION, OpType=INVOKE})
The connection succeeds.
ACU_CONN_CO
Note: No state changes occur.

Notification of a diverted inbound call (ETS 300)

When an inbound call is diverted by the network, the application is notified. An ACU_FACILITY_IN message is sent to the application using the ACU_SAPI_MGT SAPI using a dummy connection ID. This message contains an acu_ss_divert_ret_result extended data structure.


Stack
Application
The network indicates that a call originally destined for this user has been diverted elsewhere.
ACU_FACILITY_IN (sapi= ACU_SAPI_MGMT, { SS:OperationId= DIVERSION, OpType=RetRes})
Note: No state changes occur.

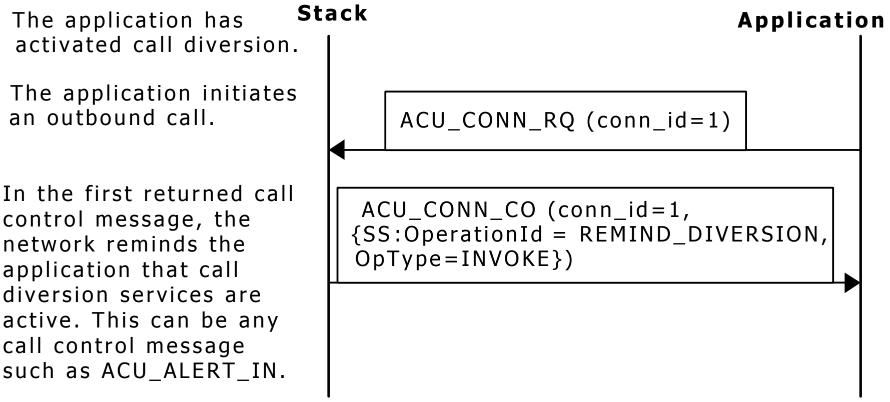
Remind diversion service (ETS 300)

If the served user activates call diversion services, whenever the served user initiates an outbound call, the remind diversion services can remind the application that call diversion services are active. If the remind diversion service is active, an acu_ss_reminder_diversion_invoke supplementary service structure is returned to the application in the first incoming call control message.

Any incoming call control message, such as ACU_CONN_CO and ACU_ALERT_IN, can carry the remind diversion message.

Remind diversion can optionally be activated when the user first subscribes to PRI services. It cannot be remotely activated or invoked by the application.

If the application activated call diversion only for calls of a specific basic service type, the remind diversion message is returned only when the served user makes an outbound call of that service type. For example, if the application activated call diversion only for teletex calls, the remind diversion message is returned only when the served user makes a teletex call.


The application has activated call diversion.
Stack
Application
The application initiates an outbound call.
ACU_CONN_RQ (conn_id=1)
In the first returned call control message, the network reminds the application that call diversion services are active. This can be any call control message such as ACU_ALERT_IN.
ACU_CONN_CO (conn_id=1, {SS:OperationId = REMIND_DIVERSION, OpType=INVOKE})

Remind diversion

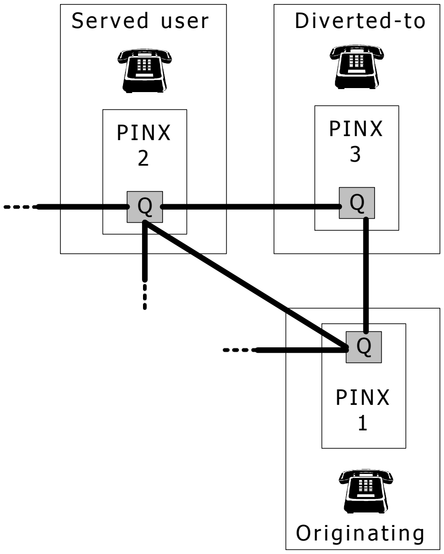
10 Call diversion services (Q.SIG)

Using call diversion services (Q.SIG)

Q.SIG describes interactions between nodes in a network. This section describes how call diversion services work from three perspectives:

- The node that makes the call (the originating node)
- The node the call is originally addressed to (the served user node)
- The node the call is diverted to (the diverted-to node)

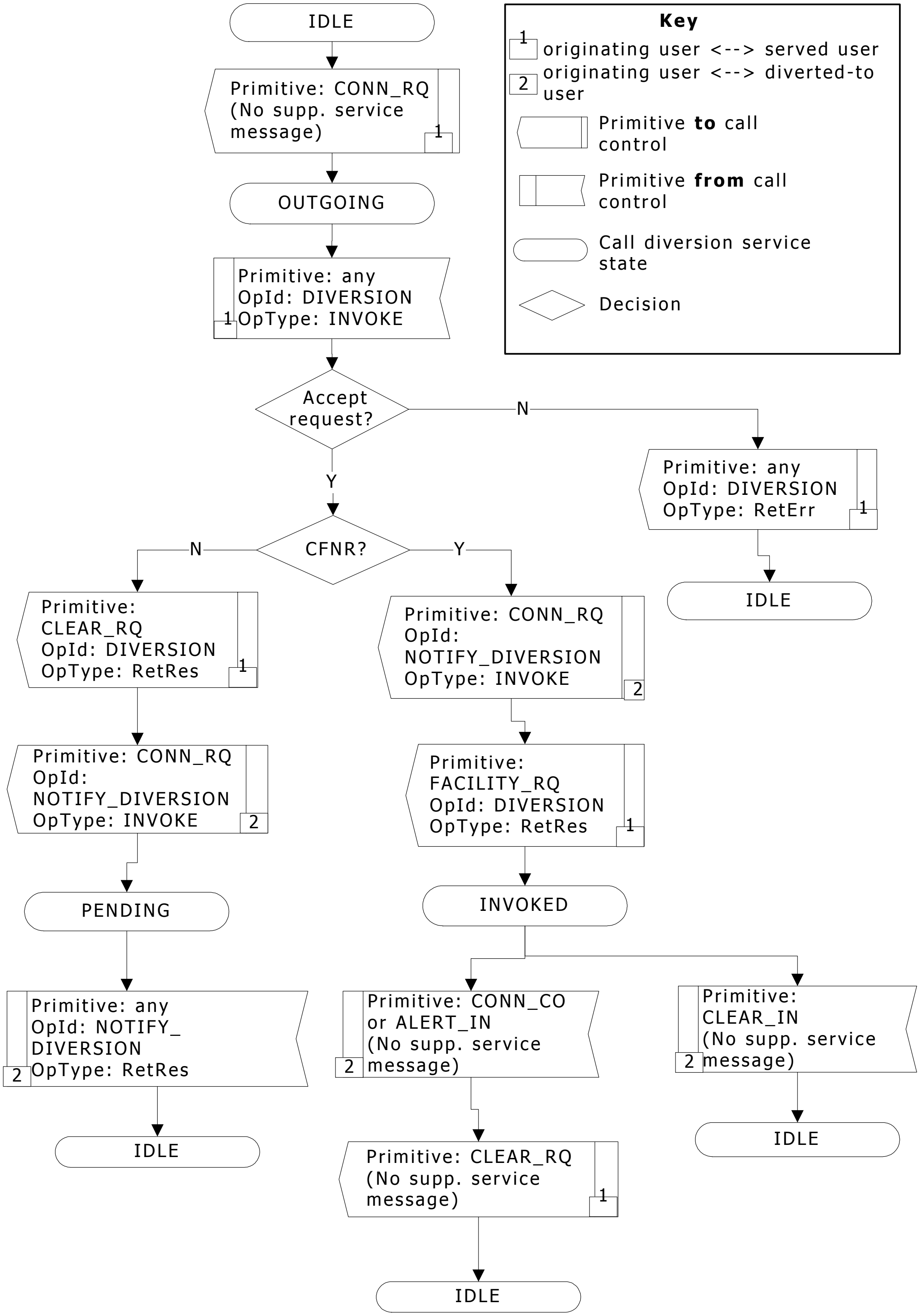
Originating, served user, and diverted-to nodes



Note: The application layer must participate in the diversion by interworking the events on the A<->B and A<->C legs of the call.

Call diversion service in the Q.SIG implementation

The following illustration shows the call diversion service in the NMS ISDN Q.SIG implementation (from the originating user's perspective):


Key
1 originating user <--> served user
2 originating user <--> diverted-to user
Primitive to call control
Primitive from call control
Call diversion service state
Decision
IDLE
Primitive: CONN_RQ (No supp. service message) 1
OUTGOING
Primitive: any OpId: DIVERSION OpType: INVOKE 1
Accept request?
N
Primitive: any OpId: DIVERSION OpType: RetErr 1
IDLE
Y
CFNR?
N
Y
Primitive: CLEAR_RQ OpId: DIVERSION OpType: RetRes 1
Primitive: CONN_RQ OpId: NOTIFY_DIVERSION OpType: INVOKE 2
PENDING
Primitive: any OpId: NOTIFY_ DIVERSION OpType: RetRes 2
IDLE
Primitive: CONN_RQ OpId: NOTIFY_DIVERSION OpType: INVOKE 2
Primitive: FACILITY_RQ OpId: DIVERSION OpType: RetRes 1
INVOKED
Primitive: CONN_CO or ALERT_IN (No supp. service message) 2
Primitive: CLEAR_RQ (No supp. service message) 1
IDLE
Primitive: CLEAR_IN (No supp. service message) 2
IDLE

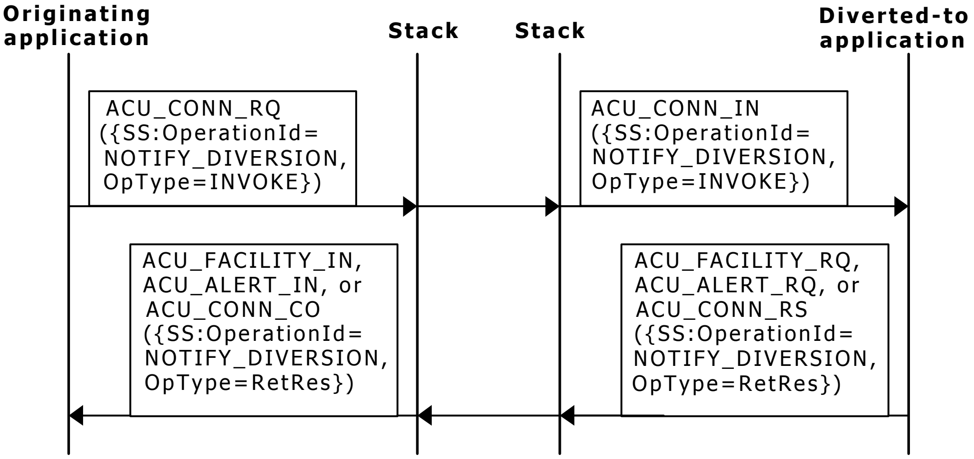
Invoking a call diversion (Q.SIG)

In the NMS ISDN Q.SIG implementation, the application at the originating node is responsible for attempting to reroute a call if it is requested to do so by the served user node's application. When the originating node offers a call to the served user node, the served user application requests the diversion by sending an acu_ss_divert_invoke extended data structure to the stack in an ACU request message.

Call diversion notification (Q.SIG)

After the alert, the originating node offers the call to the diverted-to node by sending an ACU_CONN_RQ message to the stack containing an acu_ss_notify_diversion_invoke extended data structure.

When the diverted-to node receives this request, it responds with an acu_ss_notify_diversion_ret_result extended data structure. This may be carried in an ACU_FACILITY_IN, ACU_CONN_CO, or ACU_ALERT_IN message. The message can also contain the presentation restrictions concerning the diverted-to party, as shown in the following illustration:


Originating application
Stack
Stack
Diverted-to application
ACU_CONN_RQ ({SS:OperationId= NOTIFY_DIVERSION, OpType=INVOKE})
ACU_CONN_IN ({SS:OperationId= NOTIFY_DIVERSION, OpType=INVOKE})
ACU_FACILITY_IN, ACU_ALERT_IN, or ACU_CONN_CO ({SS:OperationId= NOTIFY_DIVERSION, OpType=RetRes})
ACU_FACILITY_RQ, ACU_ALERT_RQ, or ACU_CONN_RS ({SS:OperationId= NOTIFY_DIVERSION, OpType=RetRes})

Call diversion messaging (Q.SIG)

The following illustrations show the sequence of messages passed between the three nodes in the following scenarios:

- Successful call forward - unconditional (Q.SIG) - The served user node is directed to forward calls unconditionally (CFU), and the diversion attempt succeeds.
- Failed call forward - unconditional (Q.SIG) - The served user node is directed to forward calls unconditionally (CFU), and the diversion fails because the diverted-to node refuses the call.
- Successful call forward - no response (Q.SIG) - The served user node is directed to forward calls if the called party does not respond (CFNR), and the diversion attempt succeeds.
- Failed call forward - no response (Q.SIG) - The served user node is directed to forward calls if the called party does not respond (CFNR), and the diversion attempt fails because the diverted-to node refuses the call.
- Aborted call forward - no response (Q.SIG) - The served user node is directed to forward calls if the called party does not respond (CFNR), and the diversion attempt fails because the called party picks up before the diverted-to party answers.

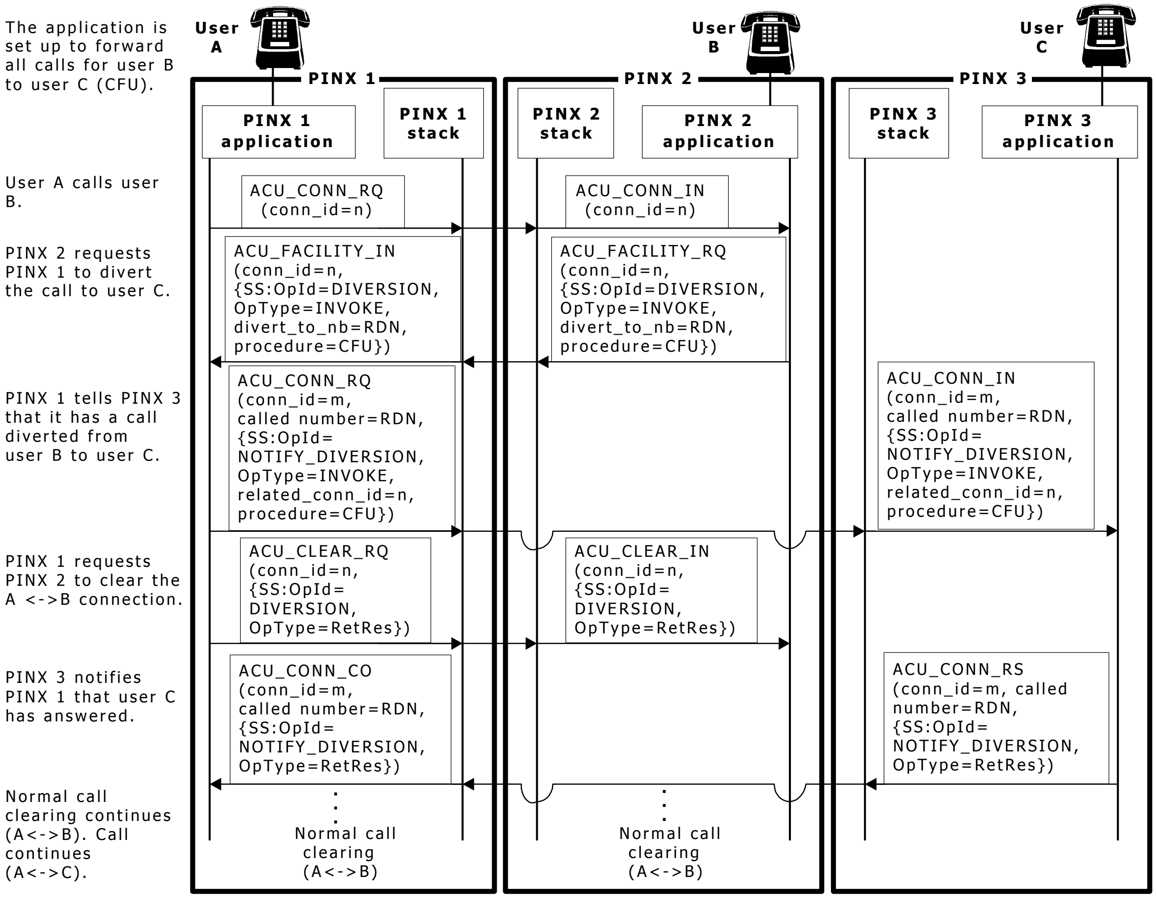
These illustrations are based on Figures C.1, C.2, and C.3 in Appendix C of ISO/IEC 13873. The ACU interface represents the interaction between the SS-DIV control entity with the user. Certain modifications are necessary to take into account that the real Q reference point is in the application, and not in the stack. Therefore, some data that would not be sent to the user is sent to the application.

Also, Figures C.1 and C.2 of the ISO/IEC specification have been combined into a single ACU interface because the diversion services are always invoked on the originating node (for example, diversion by forward switching is not supported).

Successful call forward - unconditional (Q.SIG)

The following illustration shows the message types passed between nodes when a call forward - unconditional succeeds. In this scenario, the application on PINX 2 is set up to forward all calls destined for user B to user C.

Note: This scenario is identical to call forward - busy (CFB) except for the operation IDs.


The application is set up to forward all calls for user B to user C (CFU).
User A
User B
User C
PINX 1
PINX 2
PINX 3
PINX 1 application
PINX 1 stack
PINX 2 stack
PINX 2 application
PINX 3 stack
PINX 3 application
User A calls user B.
ACU_CONN_RQ (conn_id=n)
ACU_CONN_IN (conn_id=n)
PINX 2 requests PINX 1 to divert the call to user C.
ACU_FACILITY_IN (conn_id=n, {SS:OpId=DIVERSION, OpType=INVOKE, divert_to_nb=RDN, procedure=CFU})
ACU_FACILITY_RQ (conn_id=n, {SS:OpId=DIVERSION, OpType=INVOKE, divert_to_nb=RDN, procedure=CFU})
PINX 1 tells PINX 3 that it has a call diverted from user B to user C.
ACU_CONN_RQ (conn_id=m, called number=RDN, {SS:OpId= NOTIFY_DIVERSION, OpType=INVOKE, related_conn_id=n, procedure=CFU})
ACU_CONN_IN (conn_id=m, called number=RDN, {SS:OpId= NOTIFY_DIVERSION, OpType=INVOKE, related_conn_id=n, procedure=CFU})
PINX 1 requests PINX 2 to clear the A <->B connection.
ACU_CLEAR_RQ (conn_id=n, {SS:OpId= DIVERSION, OpType=RetRes})
ACU_CLEAR_IN (conn_id=n, {SS:OpId= DIVERSION, OpType=RetRes})
PINX 3 notifies PINX 1 that user C has answered.
ACU_CONN_CO (conn_id=m, called number=RDN, {SS:OpId= NOTIFY_DIVERSION, OpType=RetRes})
ACU_CONN_RS (conn_id=m, called number=RDN, {SS:OpId= NOTIFY_DIVERSION, OpType=RetRes})
Normal call clearing continues (A<->B). Call continues (A<->C).
Normal call clearing (A<->B)
Normal call clearing (A<->B)

Failed call forward - unconditional (Q.SIG)

The following illustration shows the message types passed between nodes when a call forward - unconditional fails because the diverted-to node rejects the call or returns an error. In this scenario, the application on PINX 2 is set up to forward all calls destined for user B to user C.

Note: This scenario is identical to call forward - busy (CFB) except for the operation IDs.

The application is set up to forward all calls for user B to user C (CFU).

User A calls user B.

PINX 2 requests PINX 1 to divert the call to user C.

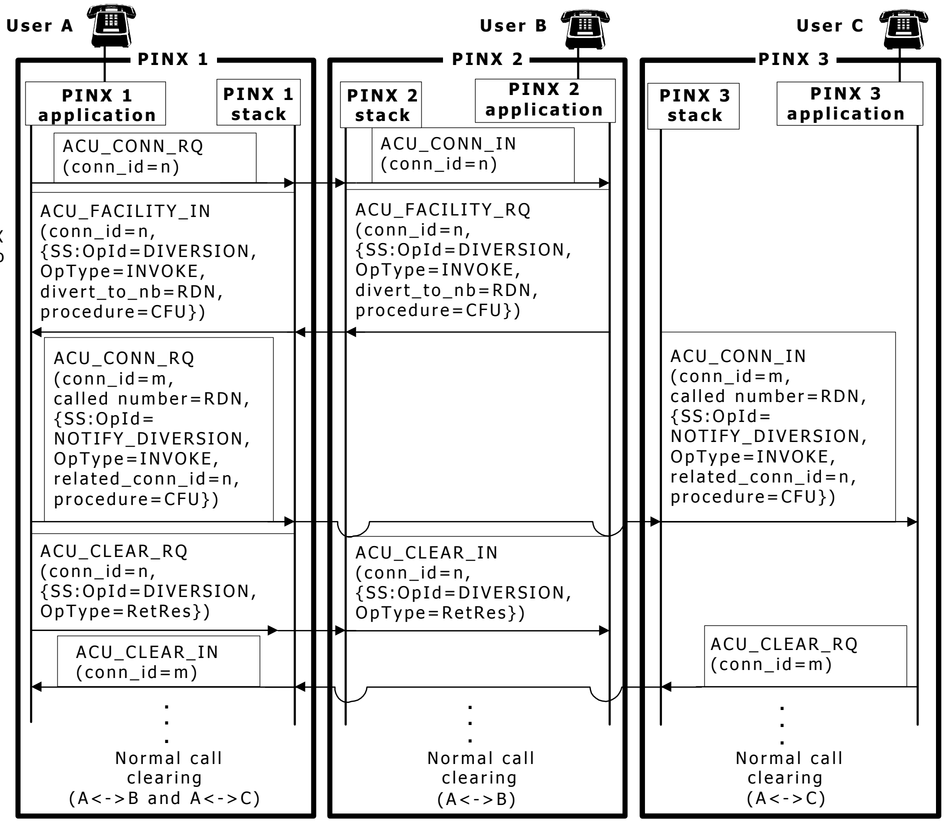
PINX 1 tells PINX 3 that it has a call diverted from user B to user C.

PINX 1 tells PINX 2 that user B has been contacted.

PINX 3 sends a clear request to PINX 1, indicating that it did not take the call.

The call is lost.

Normal call clearing continues.


User A
User B
User C
PINX 1
PINX 2
PINX 3
PINX 1 application
PINX 1 stack
PINX 2 stack
PINX 2 application
PINX 3 stack
PINX 3 application
ACU_CONN_RQ (conn_id=n)
ACU_CONN_IN (conn_id=n)
ACU_FACILITY_IN (conn_id=n, {SS:OpId=DIVERSION, OpType=INVOKE, divert_to_nb=RDN, procedure=CFU})
ACU_FACILITY_RQ (conn_id=n, {SS:OpId=DIVERSION, OpType=INVOKE, divert_to_nb=RDN, procedure=CFU})
ACU_CONN_RQ (conn_id=m, called number=RDN, {SS:OpId= NOTIFY_DIVERSION, OpType=INVOKE, related_conn_id=n, procedure=CFU})
ACU_CONN_IN (conn_id=m, called number=RDN, {SS:OpId= NOTIFY_DIVERSION, OpType=INVOKE, related_conn_id=n, procedure=CFU})
ACU_CLEAR_RQ (conn_id=n, {SS:OpId=DIVERSION, OpType=RetRes})
ACU_CLEAR_IN (conn_id=n, {SS:OpId=DIVERSION, OpType=RetRes})
ACU_CLEAR_IN (conn_id=m)
ACU_CLEAR_RQ (conn_id=m)
Normal call clearing (A<->B and A<->C)
Normal call clearing (A<->B)
Normal call clearing (A<->C)

Successful call forward - no response (Q.SIG)

The following illustration shows the message types passed between nodes when a call forward - no response succeeds.

In this scenario, the application on PINX 2 is set up to forward all calls destined for user B to user C if user B does not respond within a period of time.

The application is set up to forward all calls for user B to user C, if user B does not respond (CFNR).

User A calls user B.

PINX 2 tells PINX 1 that user B's phone is ringing.

The call attempt times out.

PINX 2 invokes CFNR by telling PINX 1 that user B is not responding, but to try user C.

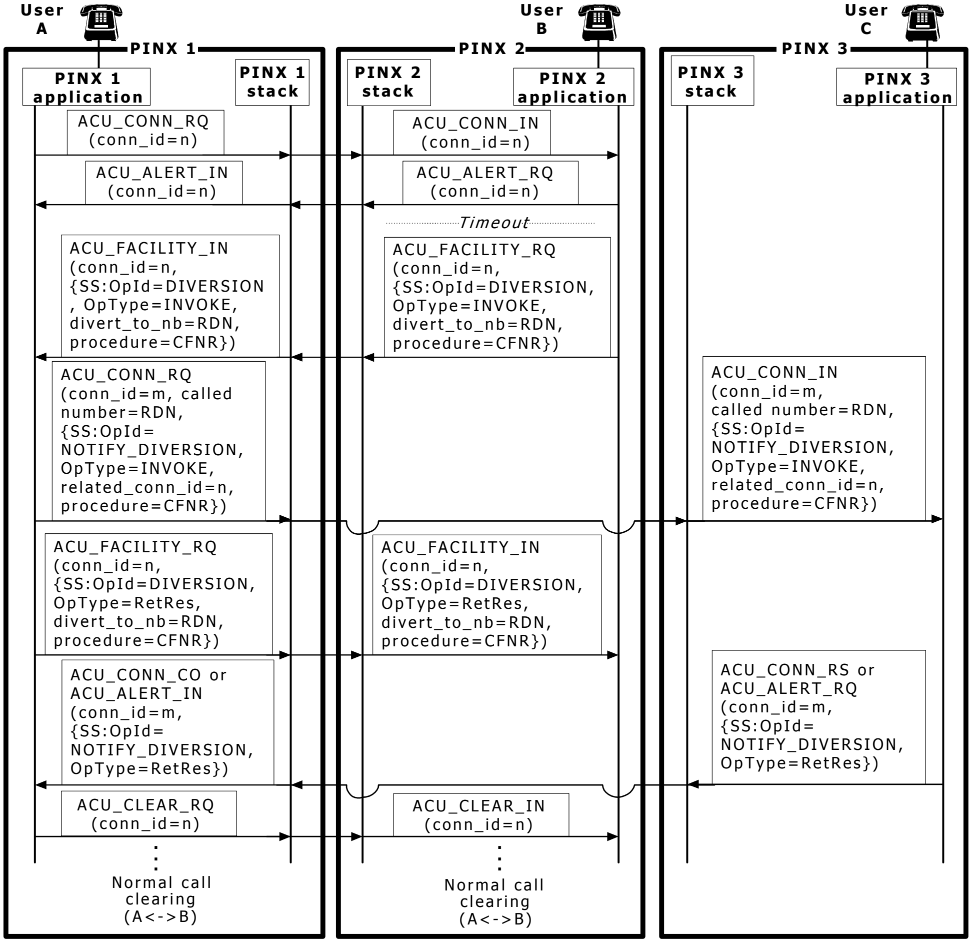
PINX 1 tells PINX 3 that it has a call diverted from user B to user C.

PINX 1 tells PINX 2 that user B has been contacted.

PINX 3 tells PINX 1 that user C has answered the call (or his phone is ringing).

PINX 1 need not wait for user C to answer anymore, so it begins Clearing the C<->D leg.

Normal call clearing continues.


User A
User B
User C
PINX 1
PINX 2
PINX 3
PINX 1 application
PINX 1 stack
PINX 2 stack
PINX 2 application
PINX 3 stack
PINX 3 application
ACU_CONN_RQ (conn_id=n)
ACU_CONN_IN (conn_id=n)
ACU_ALERT_IN (conn_id=n)
ACU_ALERT_RQ (conn_id=n)
Timeout
ACU_FACILITY_IN (conn_id=n, {SS:OpId=DIVERSION , OpType=INVOKE, divert_to_nb=RDN, procedure=CFNR})
ACU_FACILITY_RQ (conn_id=n, {SS:OpId=DIVERSION, OpType=INVOKE, divert_to_nb=RDN, procedure=CFNR})
ACU_CONN_RQ (conn_id=m, called number=RDN, {SS:OpId= NOTIFY_DIVERSION, OpType=INVOKE, related_conn_id=n, procedure=CFNR})
ACU_CONN_IN (conn_id=m, called number=RDN, {SS:OpId= NOTIFY_DIVERSION, OpType=INVOKE, related_conn_id=n, procedure=CFNR})
ACU_FACILITY_RQ (conn_id=n, {SS:OpId=DIVERSION, OpType=RetRes, divert_to_nb=RDN, procedure=CFNR})
ACU_FACILITY_IN (conn_id=n, {SS:OpId=DIVERSION, OpType=RetRes, divert_to_nb=RDN, procedure=CFNR})
ACU_CONN_CO or ACU_ALERT_IN (conn_id=m, {SS:OpId= NOTIFY_DIVERSION, OpType=RetRes})
ACU_CONN_RS or ACU_ALERT_RQ (conn_id=m, {SS:OpId= NOTIFY_DIVERSION, OpType=RetRes})
ACU_CLEAR_RQ (conn_id=n)
ACU_CLEAR_IN (conn_id=n)
Normal call clearing (A<->B)
Normal call clearing (A<->B)

Failed call forward - no response (Q.SIG)

The following illustration shows the message types passed between nodes when a call forward - no response fails because the diverted-to node rejects the call or returns an error.

In this scenario, the application on PINX 2 is set up to forward all calls destined for user B to user C if user B does not respond within a period of time.

The application is set up to forward all calls for user B to user C, if user B does not respond (CFNR).

User A calls user B.

PINX 2 tells PINX 1 that user B's phone is ringing. The call attempt times out.

The call attempt times out.

PINX 2 invokes CFNR by telling PINX 1 that user B is not responding, and to try user C.

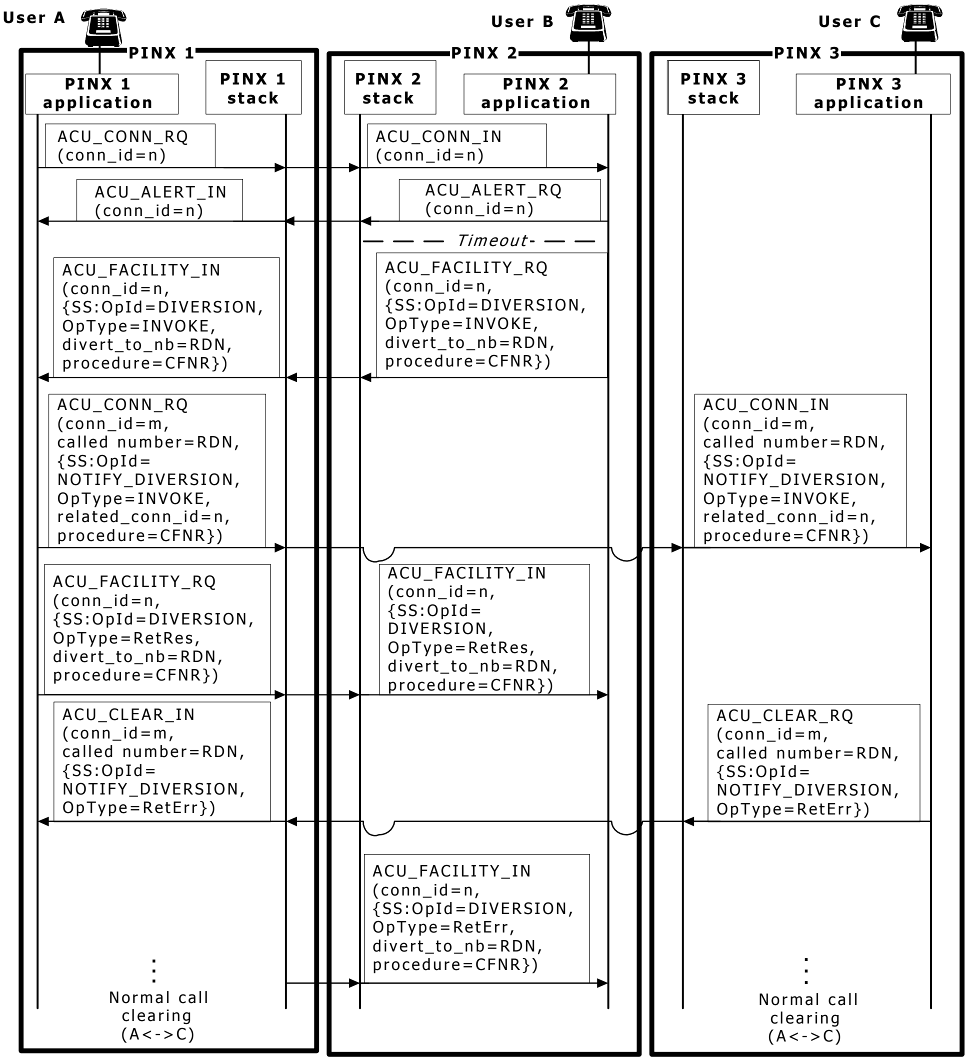
PINX 1 tells PINX 3 that it has a call diverted from user B to user C.

PINX 1 tells PINX 2 that user B has been contacted.

PINX 3 sends an error message to PINX 1.

PINX 1's stack tells PINX 2 that user B could not be reached.

PINX 2 may now try again, try another diversion, or clear the call.


User A
User B
User C
PINX 1
PINX 2
PINX 3
PINX 1 application
PINX 1 stack
PINX 2 stack
PINX 2 application
PINX 3 stack
PINX 3 application
ACU_CONN_RQ (conn_id=n)
ACU_CONN_IN (conn_id=n)
ACU_ALERT_IN (conn_id=n)
ACU_ALERT_RQ (conn_id=n)
Timeout-
ACU_FACILITY_IN (conn_id=n, {SS:OpId=DIVERSION, OpType=INVOKE, divert_to_nb=RDN, procedure=CFNR})
ACU_FACILITY_RQ (conn_id=n, {SS:OpId=DIVERSION, OpType=INVOKE, divert_to_nb=RDN, procedure=CFNR})
ACU_CONN_RQ (conn_id=m, called number=RDN, {SS:OpId= NOTIFY_DIVERSION, OpType=INVOKE, related_conn_id=n, procedure=CFNR})
ACU_CONN_IN (conn_id=m, called number=RDN, {SS:OpId= NOTIFY_DIVERSION, OpType=INVOKE, related_conn_id=n, procedure=CFNR})
ACU_FACILITY_RQ (conn_id=n, {SS:OpId=DIVERSION, OpType=RetRes, divert_to_nb=RDN, procedure=CFNR})
ACU_FACILITY_IN (conn_id=n, {SS:OpId= DIVERSION, OpType=RetRes, divert_to_nb=RDN, procedure=CFNR})
ACU_CLEAR_IN (conn_id=m, called number=RDN, {SS:OpId= NOTIFY_DIVERSION, OpType=RetErr})
ACU_CLEAR_RQ (conn_id=m, called number=RDN, {SS:OpId= NOTIFY_DIVERSION, OpType=RetErr})
ACU_FACILITY_IN (conn_id=n, {SS:OpId=DIVERSION, OpType=RetErr, divert_to_nb=RDN, procedure=CFNR})
Normal call clearing (A<->C)
Normal call clearing (A<->C)

Aborted call forward - no response (Q.SIG)

The following illustration shows the message types passed between nodes when a call forward - no response is aborted during the forwarding attempt because the served user node senses that the user to whom the call was originally destined has answered.

In this scenario, the application on PINX 2 is set up to forward all calls destined for user B to user C if user B does not respond within a period of time.

The application is set up to forward all calls for user B to user C, if user B does not respond (CFNR).

User A calls user B.

PINX 2 tells PINX 1 that user B's phone is ringing.

The call attempt times out.

PINX 2 invokes CFNR by telling PINX 1 that user B is not responding, and to try user C.

PINX 1 tells PINX 3 that it has a call diverted from user B to user C.

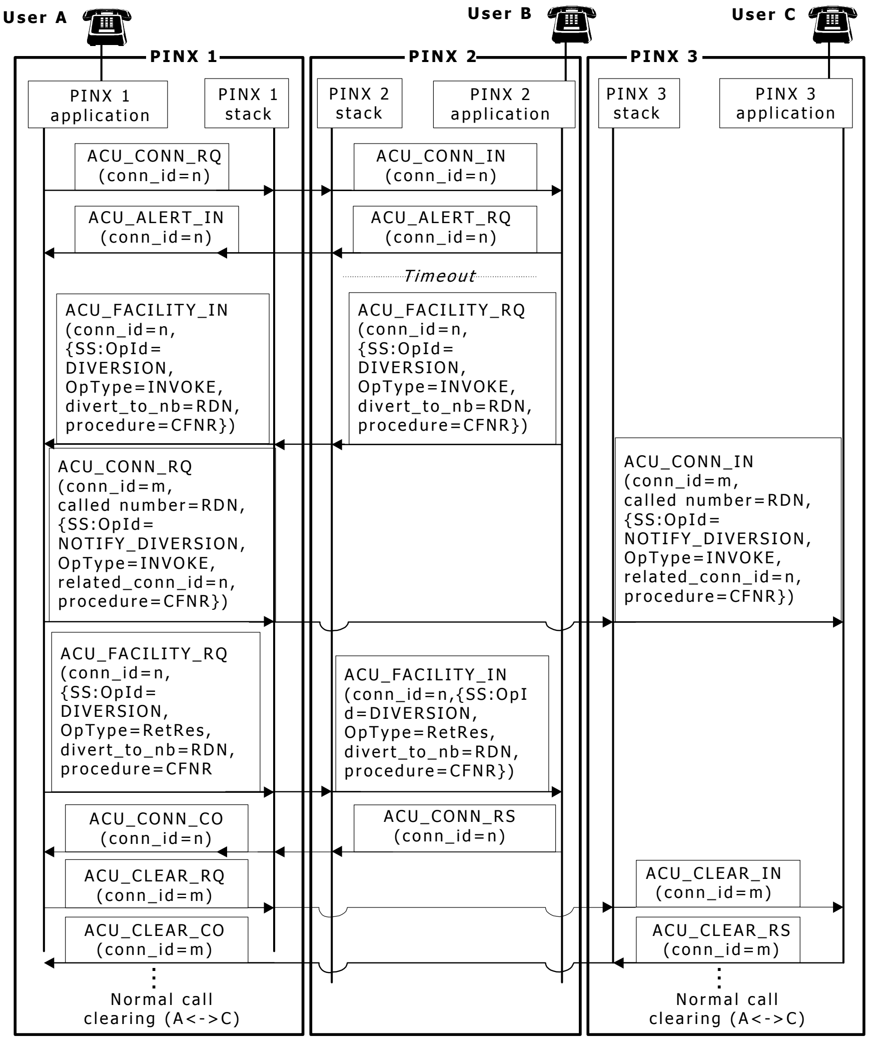
PINX 1 tells PINX 2 that user B has been contacted.

User B answers before C's phone starts ringing. PINX 2 tells PINX 1 that user B has answered.

PINX 1 abandons (clears) the attempt to divert the call (tells PINX 3 to clear it).

PINX 3 sends a clear confirmation to PINX 1.

Normal call clearing continues (A<->C).


User A
User B
User C
PINX 1
PINX 2
PINX 3
PINX 1 application
PINX 1 stack
PINX 2 stack
PINX 2 application
PINX 3 stack
PINX 3 application
ACU_CONN_RQ (conn_id=n)
ACU_CONN_IN (conn_id=n)
ACU_ALERT_IN (conn_id=n)
ACU_ALERT_RQ (conn_id=n)
Timeout
ACU_FACILITY_IN (conn_id=n, {SS:OpId= DIVERSION, OpType=INVOKE, divert_to_nb=RDN, procedure=CFNR})
ACU_FACILITY_RQ (conn_id=n, {SS:OpId= DIVERSION, OpType=INVOKE, divert_to_nb=RDN, procedure=CFNR})
ACU_CONN_RQ (conn_id=m, called number=RDN, {SS:OpId= NOTIFY_DIVERSION, OpType=INVOKE, related_conn_id=n, procedure=CFNR})
ACU_CONN_IN (conn_id=m, called number=RDN, {SS:OpId= NOTIFY_DIVERSION, OpType=INVOKE, related_conn_id=n, procedure=CFNR})
ACU_FACILITY_RQ (conn_id=n, {SS:OpId= DIVERSION, OpType=RetRes, divert_to_nb=RDN, procedure=CFNR
ACU_FACILITY_IN (conn_id=n,{SS:OpId=DIVERSION, OpType=RetRes, divert_to_nb=RDN, procedure=CFNR})
ACU_CONN_CO (conn_id=n)
ACU_CONN_RS (conn_id=n)
ACU_CLEAR_RQ (conn_id=m)
ACU_CLEAR_IN (conn_id=m)
ACU_CLEAR_CO (conn_id=m)
ACU_CLEAR_RS (conn_id=m)
Normal call clearing (A<->C)
Normal call clearing (A<->C)

Unsuccessful diversion (Q.SIG)

If a diversion is unsuccessful, the diverted-to node sends the originating node an ACU request message containing an acu_ss_divert_ret_error extended data structure.

11 Call deflection

Call deflection overview

Call deflection services allow calls presented to the served user to be rerouted to a different address. Call deflection is only applicable to user/network type interfaces such as ETS 300, and not to peer-to-peer interfaces such as Q.SIG.

An ETS 300 served user application can use the call deflection service in either of two ways:

- It can activate the call deflection service. When activated, call deflection takes place in the stack. From then until the service is deactivated, all calls on a specific NAI destined for the served user are rerouted to another user.
- It can invoke the call deflection service on a call-by-call basis.

Regardless of which service is used, the diverted-to user is always notified of a forwarded call using a notify diversion operation.

Activating call deflection for all calls on an NAI

Activation of the call deflection service has only local significance (for example, no signaling information is exchanged with the remote peer). It applies only to activation/deactivation on an NAI basis for all served users.

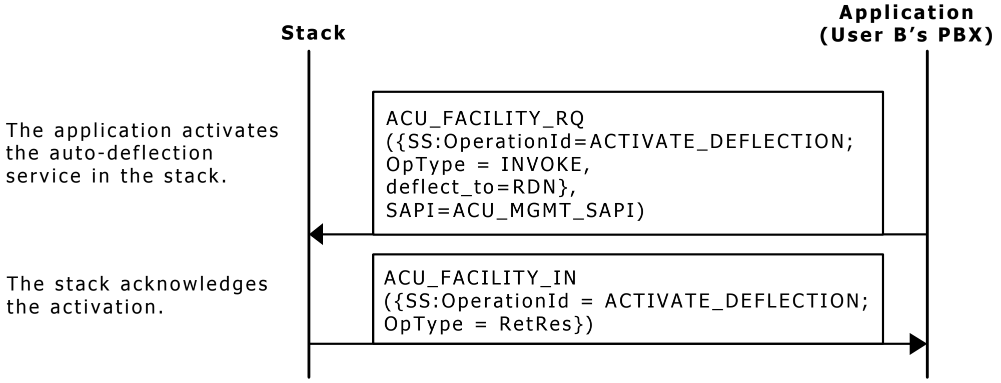
To activate call deflection for an NAI, the application sends an ACU request message to the stack to the ACU_MGMT_SAPI containing an acu_ss_activate_deflect_invoke extended data structure.

Successful call deflection activation

If call deflection is successfully activated, the stack returns an ACU indication message to the application containing an acu_ss_activate_deflect_ret_result extended data structure.

(From the perspective of User B's PBX) A is calling B.

The PBX application decides that calls directed to any served user should be deflected back to A.


Stack
Application (User B's PBX)
ACU_FACILITY_RQ ({SS:OperationId=ACTIVATE_DEFLECTION; OpType = INVOKE, deflect_to=RDN}, SAPI=ACU_MGMT_SAPI)
ACU_FACILITY_IN ({SS:OperationId = ACTIVATE_DEFLECTION; OpType = RetRes})

The application activates the auto-deflection service in the stack.

The stack acknowledges the activation.

Activation of call deflection

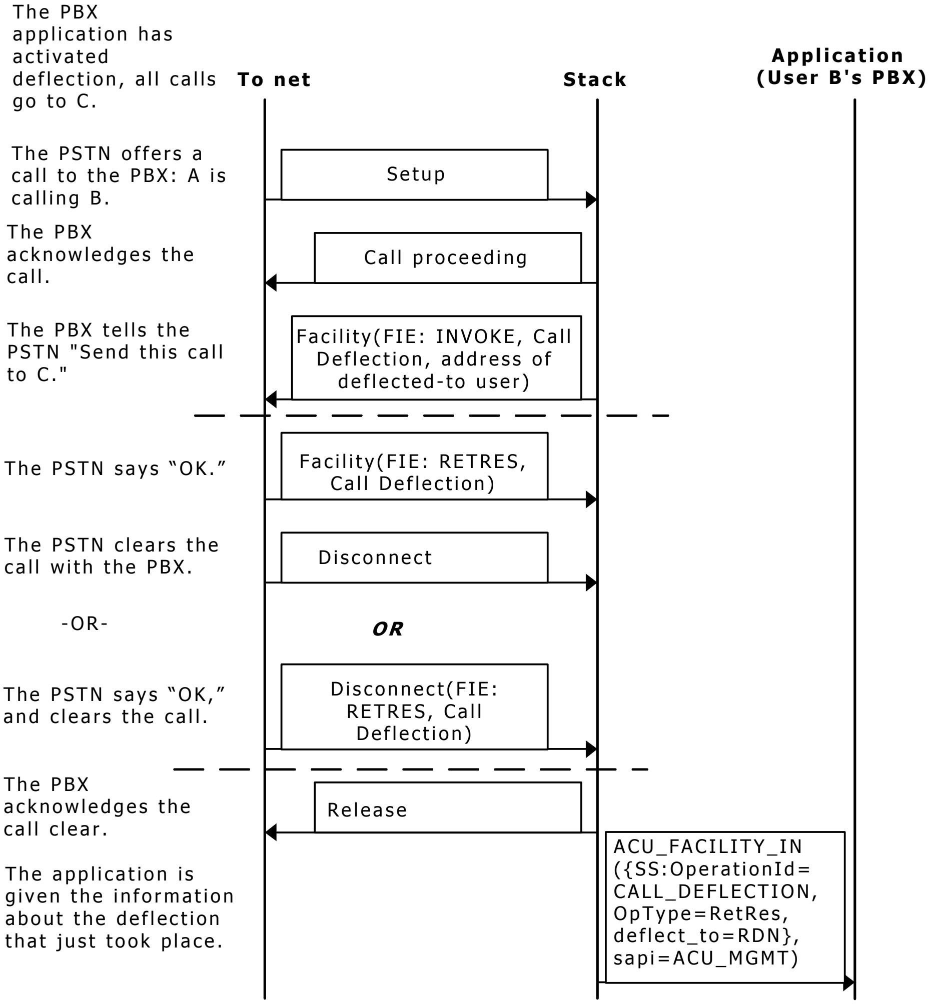
Unsuccessful call deflection activation

If call deflection is not successfully activated, the stack returns an ACU indication message to the application containing an acu_ss_reject extended data structure.

Notification of a call deflection

If call deflection is activated for all calls on an NAI, the application receives a notification when a call is deflected through the ACU_MGMT_SAPI. The stack returns an ACU indication message to the application containing an acu_ss_deflect_ret_result extended data structure.

The following illustration shows the exchange of messages that takes place when call deflection is activated for an NAI. In this scenario, user A is calling user B, and user B (for example, PBX) deflects the call to user C.


The PBX application has activated deflection, all calls go to C.
To net
Stack
Application (User B's PBX)
The PSTN offers a call to the PBX: A is calling B.
Setup
The PBX acknowledges the call.
Call proceeding
The PBX tells the PSTN "Send this call to C."
Facility(FIE: INVOKE, Call Deflection, address of deflected-to user)
The PSTN says "OK."
Facility(FIE: RETRES, Call Deflection)
The PSTN clears the call with the PBX.
Disconnect
-OR-
OR
The PSTN says "OK," and clears the call.
Disconnect(FIE: RETRES, Call Deflection)
The PBX acknowledges the call clear.
Release
The application is given the information about the deflection that just took place.
ACU_FACILITY_IN ({SS:OperationId= CALL_DEFLECTION, OpType=RetRes, deflect_to=RDN}, sapi=ACU_MGMT)

Deactivating call deflection for all calls on an NAI

To deactivate call deflection for an NAI, the application sends an ACU request message to the stack to the ACU_MGMT_SAPI containing an acu_ss_deactivate_deflect_invoke extended data structure.

Successful call deflection deactivation

If call deflection is successfully deactivated, the stack returns an ACU indication message to the application containing an acu_ss_deactivate_deflect_ret_result extended data structure.

Unsuccessful call deflection deactivation

If call deflection is not successfully deactivated, the stack returns an ACU indication message to the application containing an acu_ss_reject extended data structure.

Invoking call deflection for a single call

The call deflection service can be invoked by the served user on an incoming call if the call state is WAIT_INCOMING.

If the stack is configured to automatically send the ALERT on incoming calls, the deflection will fail due to invalid states at the network side.

Note: The ALERT is sent by default on the user side when no structure is passed to **isdnStartProtocol**. To change this behavior, change the settings of the CC_VOICE_ALERT_RQ and CC_DATA_ALERT_RQ bits in the in_calls_behavior substructure (described in the *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual*). You can cause the stack to send a PROC RQ instead by setting the CC_SEND_CALL_PROC_RQ bit.

To invoke call deflection for a call, the application sends an ACU request message to the stack containing an acu_ss_deflect_invoke extended data structure.

Successful call deflection invocation

If call deflection is successfully invoked for the call, the stack returns an ACU indication message to the stack containing an acu_ss_deflect_ret_result extended data structure.

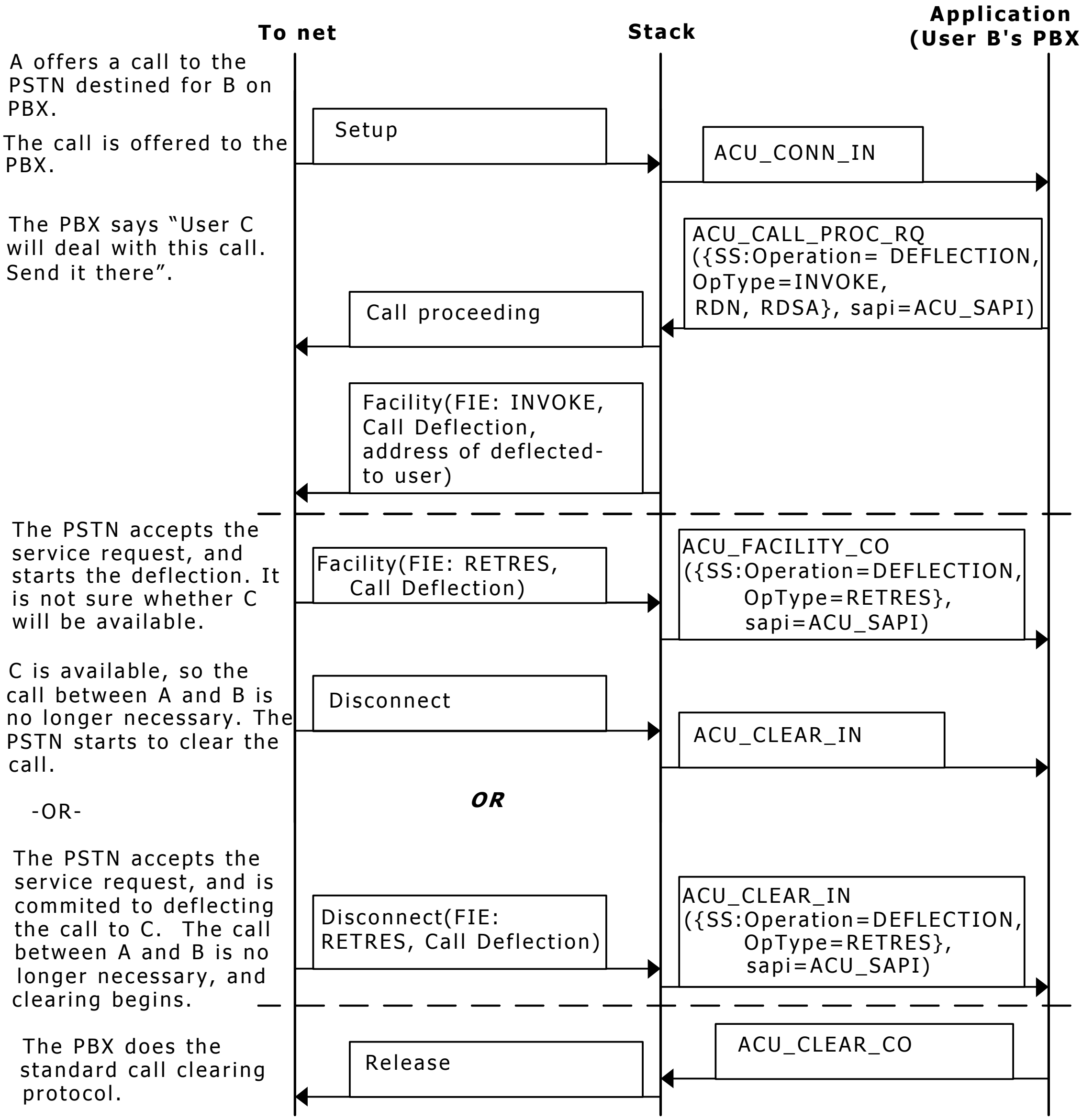
Unsuccessful call deflection invocation

If call deflection is not successfully invoked for the call, the stack returns an ACU indication message to the application containing an acu_ss_deflect_ret_error extended data structure.

Call deflection illustration

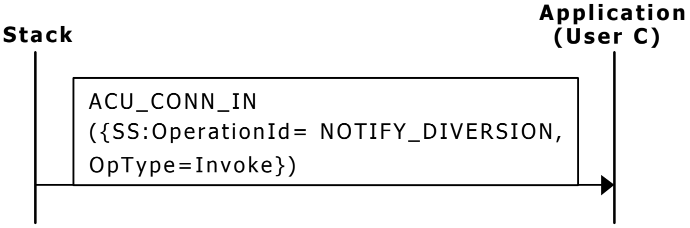
The following illustration shows the exchange of messages that takes place when call deflection is invoked for a call. In this scenario, user A is calling user B, and user B (for example, PBX) deflects the call to user C.

Note: The following illustration is representative of exchanges, and is not intended to be complete. Many factors impact the message flow, including the subscription option configurations at the network side of the interface.


To net
Stack
Application (User B's PBX)
A offers a call to the PSTN destined for B on PBX.
The call is offered to the PBX.
Setup
ACU_CONN_IN
The PBX says "User C will deal with this call. Send it there".
ACU_CALL_PROC_RQ ({SS:Operation= DEFLECTION, OpType=INVOKE, RDN, RDSA}, sapi=ACU_SAPI)
Call proceeding
Facility(FIE: INVOKE, Call Deflection, address of deflected-to user)
The PSTN accepts the service request, and starts the deflection. It is not sure whether C will be available.
Facility(FIE: RETRES, Call Deflection)
ACU_FACILITY_CO ({SS:Operation=DEFLECTION, OpType=RETRES}, sapi=ACU_SAPI)
C is available, so the call between A and B is no longer necessary. The PSTN starts to clear the call.
Disconnect
ACU_CLEAR_IN
-OR-
OR
The PSTN accepts the service request, and is commited to deflecting the call to C. The call between A and B is no longer necessary, and clearing begins.
Disconnect(FIE: RETRES, Call Deflection)
ACU_CLEAR_IN ({SS:Operation=DEFLECTION, OpType=RETRES}, sapi=ACU_SAPI)
The PBX does the standard call clearing protocol.
Release
ACU_CLEAR_CO

Deflection notification at deflected-to user

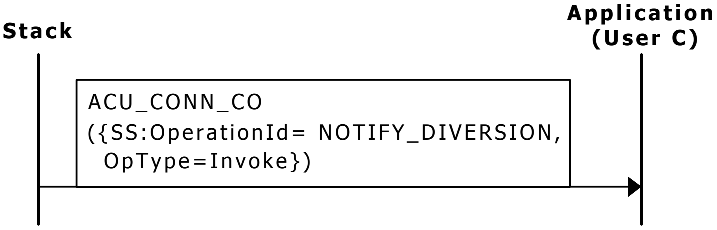
When the deflected-to user receives a connection indication, the user is notified that the call is forwarded. An ACU indication message is sent containing an acu_ss_notify_diversion_invoke extended data structure.



Connection request at deflected-to user

Deflection notification at originating user

The originating user is notified that a call is forwarded. An ACU indication or confirmation message is sent (such as ACU_CONN_CO, ACU_ALERT_IN, a PROGRESS message, or a FACILITY message) containing an acu_ss_notify_diversion_invoke extended data structure.



Connection request at originating user

12 Advice of charge services

Advice of charge services overview

This section describes how to use the NMS ISDN advice of charge (AOC) services, available within the ETS 300 variant.

AOC services provide users with a way of tracking the costs of a specific call, in real time. Three AOC services are available, each of which causes AOC information to be returned at a different point in the call:

- AOC at start of call (AOC-S). With AOC-S, the user side is notified of the cost of a call when the call is started. This information can take the form of a flat rate currency cost, the cost on a time basis, or the cost based on pre-arranged item numbers.
- AOC during the call (AOC-D). AOC-D provides the user with information about the cost of the call during the call. For example, a subtotal of the cost could be sent to the user on an interval basis.
- AOC at the end of the call (AOC-E). AOC-E provides the user side with the total cost of the call, at the time the call is cleared (or later).

Invocation of the AOC service is performed by the originating node, on a call-by-call basis. Once AOC is invoked, the originating node receives charging information using supplementary service data structures.

Alternatively, as a subscription option, the interface can be configured to provide AOC on all calls. In this case, the originating user side does not need to invoke the service. However, the NAI must be configured locally with the AOC subscription options (see *Configuring the NAI for AOC subscription services* on page 105).

Note: The basic call signaling over the network is affected by AOC services. In call clearing cases, the call clearing procedure is delayed until AOC signaling is completed.

AOC services are not available under Q.SIG.

Invoking advice of charge services

If AOC services are not provided using a subscription service, they must be invoked to be used. Invocation of the services at the originating node is accomplished on a per-call basis. To invoke AOC services, an ACU_CONN_RQ primitive is sent to the stack, containing an acu_ss_aoc_request_invoke extended data structure.

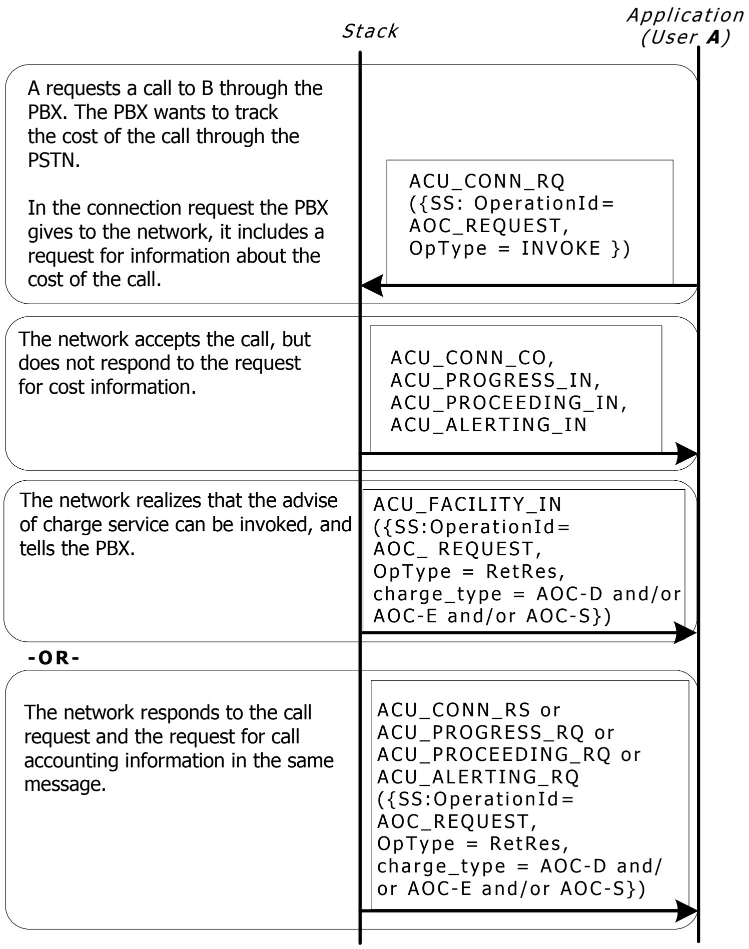
Successful AOC invocation

If AOC services are successfully invoked, an acu_ss_aoc_request_ret_result extended data structure is returned in an ACU indication or confirmation message.

The application may receive one acu_ss_aoc_request_ret_result message indicating the activation of all AOC services, or one for each service. If the application receives a message indicating that not all requested services are activated, this does not necessarily imply an error. It may mean that results are not yet available for the other services. Subsequent messages contain the activation results for these services.

Unsuccessful AOC invocation

If AOC services are not successfully invoked, an acu_ss_aoc_request_ret_error extended data structure is returned in an ACU indication or confirmation message.


Stack
Application (User A)
A requests a call to B through the PBX. The PBX wants to track the cost of the call through the PSTN.
In the connection request the PBX gives to the network, it includes a request for information about the cost of the call.
ACU_CONN_RQ ({SS: OperationId= AOC_REQUEST, OpType = INVOKE })
The network accepts the call, but does not respond to the request for cost information.
ACU_CONN_CO, ACU_PROGRESS_IN, ACU_PROCEEDING_IN, ACU_ALERTING_IN
The network realizes that the advise of charge service can be invoked, and tells the PBX.
ACU_FACILITY_IN ({SS:OperationId= AOC_ REQUEST, OpType = RetRes, charge_type = AOC-D and/or AOC-E and/or AOC-S})
-OR-
The network responds to the call request and the request for call accounting information in the same message.
ACU_CONN_RS or ACU_PROGRESS_RQ or ACU_PROCEEDING_RQ or ACU_ALERTING_RQ ({SS:OperationId= AOC_REQUEST, OpType = RetRes, charge_type = AOC-D and/ or AOC-E and/or AOC-S})

AOC request procedure during call establishment

Receiving AOC data

This topic describes:

- Advice of charge - start of call (AOC-S)
- Advice of charge - during call (AOC-D)
- Advice of charge - end of call (AOC-E)
- AOC and explicit call transfer (ECT) services
- AOC and call deflection services

When AOC data is available, the application receives an ACU indication or confirmation message containing an acu_ss_aoc_inform_invoke extended data structure.

Charging information is delivered to the application using the extended service data area of an ACU primitive during the active part of the call.

Advice of charge - start of call (AOC-S)

If AOC-S is active, the aoc_type field in acu_ss_aoc_inform_invoke is set to ACU_SS_AOC_TYPE_AOC_S_INFORM. The tAcuSSAocSInform data structure and substructures carry advice of charge data.

Different charging rates are associated with each of the charged item types. If a charged item is not included, the default value is interpreted as free of charge.

Charged item	**Included**	**Possible charging rate values**
Basic communication	Always.	• Price per time unit and time unit. • Flat rate (a fixed currency value per event). • Free of charge. • Special charging code. • Not available.
Call attempt	Only in the initial charging information sent to the served user.	• Flat rate (a fixed currency value per event). • Free of charge. • Special charging code. • Not available.
Call setup	Only in the initial charging information sent to the served user.	• Flat rate (a fixed currency value per event). • Free of charge. • Special charging code. • Not available.

Charged item	Included	Possible charging rate values
Operation of supplementary services	Only if the served user requested a supplementary service.	• Price per time unit and time unit. • Flat rate (a fixed currency value per event). • Free of charge. • Special charging code. • Not available.
User-to-user information transfer	Only if the served user requested the user-to-user signaling supplementary service.	• Price per volume unit and volume unit. • Flat rate (a fixed currency value per event). • Free of charge. • Special charging code. • Not available.

The charging rate is specified in the charging_rate field in the tAcuSSAocSInform structure. Depending upon its setting, one of the following data structures contains additional data:

- tAcuSSAocDuration: provides the currency value for a particular time unit, and the length of the time unit.
- tAcuSSAocVolume: provides the currency value for a particular volume unit, and the length of the volume unit.
- tAcuSSAocSpecific: provides a specific currency value.

The following table lists the charging_rate field setting and associated data structure for each charging rate type:

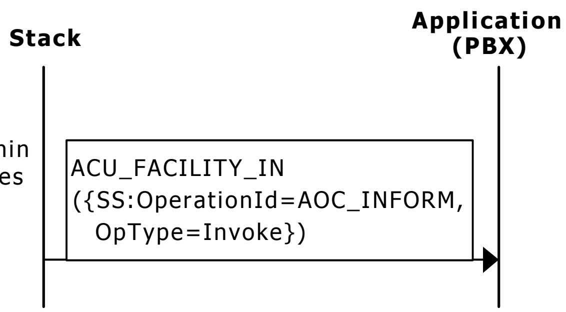
Rate type	charging_rate field setting	Associated data structure	Structure description
Price per volume unit and volume unit	VOLUME	tAcuSSAocVolume	Currency value for a particular volume unit, together with the length of the volume unit.
Price per time unit and time unit	DURATION	tAcuSSAocDuration	Currency value for a particular time unit, together with the length of the time unit.
Flat rate	SPECIFIC	tAcuSSAocSpecific	A fixed currency value per event. rate_type in this structure is set to FLAT_RATE.
Free of charge	SPECIFIC	tAcuSSAocSpecific	A fixed currency value per event. rate_type in this structure is set to FREE_OF_CHARGE or FREE_OF_CHARGE_FROM_BEGINNING.
Special charging code	SPECIFIC	tAcuSSAocSpecific	A fixed currency value per event. rate_type in this structure is set to SPECIAL_CHARGING.
Not available	SPECIFIC	tAcuSSAocSpecific	rate_type in this structure is set to NOT_AVAIL.

Advice of charge - during call (AOC-D)

If AOC-D is active, the aoc_type field in the acu_ss_aoc_inform_invoke structure is set to ACU_SS_AOC_TYPE_AOC_D_INFORM. The tAcuSSAocDInform data structure and substructures carry advice of charge data.

Advice of charge - end of call (AOC-E)

If AOC-E is active, the aoc_type field in the acu_ss_aoc_inform_invoke structure is set to ACU_SS_AOC_TYPE_AOC_E_INFORM. The tAcuSSAocEInform data structure and substructures carry advice of charge data.


Stack
Application (PBX)
ACU_FACILITY_IN ({SS:OperationId=AOC_INFORM, OpType=Invoke})

A and B are in a conversation, and the network has accepted the PBX request for the AOC service.

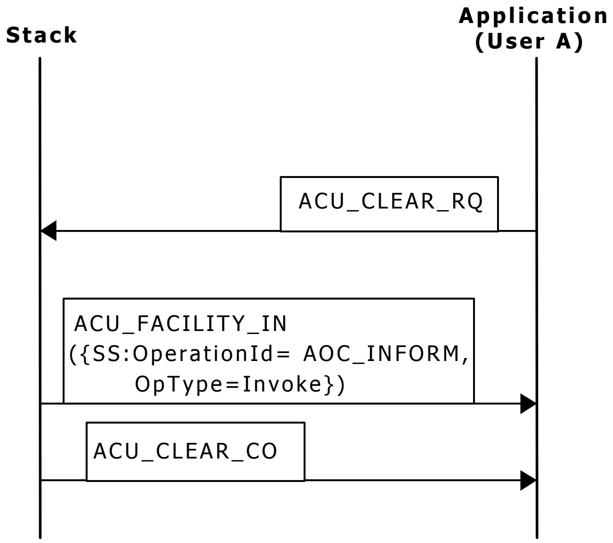
An event happens within the network that causes charge information to change or become available. The PBX is informed.

Transfer of charging information during the active state of a call

A and B are in a call, and A starts to clear it. The network has invoked AOC service on this call.

The PBX signals the network that the call between A and B can be cleared.

During the call clearing phase, the network will send final charging information to the PBX.

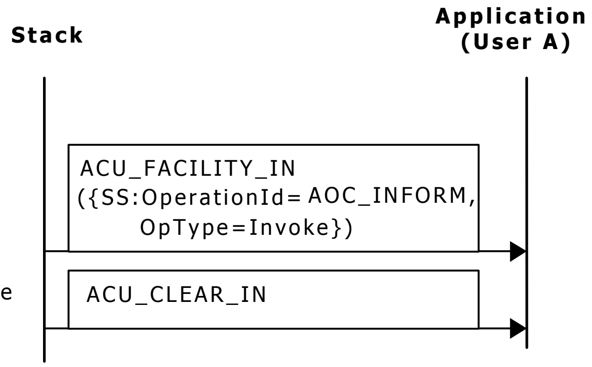

Stack
Application (User A)
ACU_CLEAR_RQ
ACU_FACILITY_IN ({SS:OperationId= AOC_INFORM, OpType=Invoke})
ACU_CLEAR_CO

Note: The application may include AOC_INFORM in the clear request rather than use a separate primitive.

Transfer of charging information during the call clearing phase, clearing initiated by the calling user

Call is active between A and B, but B has initiated clearing. The network has activated the AOC service on this call.

During the clearing phase, the final AOC is sent to the PBX.


Stack
Application (User A)
ACU_FACILITY_IN ({SS:OperationId= AOC_INFORM, OpType=Invoke})
ACU_CLEAR_IN

Note: The stack may include AOC_INFORM in the clear indication rather than use a separate primitive.

Transfer of charging information during the call clearing phase, clearing initiated by the called user

AOC and explicit call transfer (ECT) services

AOC interacts with ECT. This section describes how the services interact.

AOC-S and ECT

If AOC-S is active at the time the served user invokes ECT, AOC-S is stopped, and the application may get advice of charge prior to the clearing of the call.

AOC-D and ECT

If AOC-D is active at the time the served user invokes ECT, the application receives advice of charge information prior to the clearing of the call, and the service is stopped.

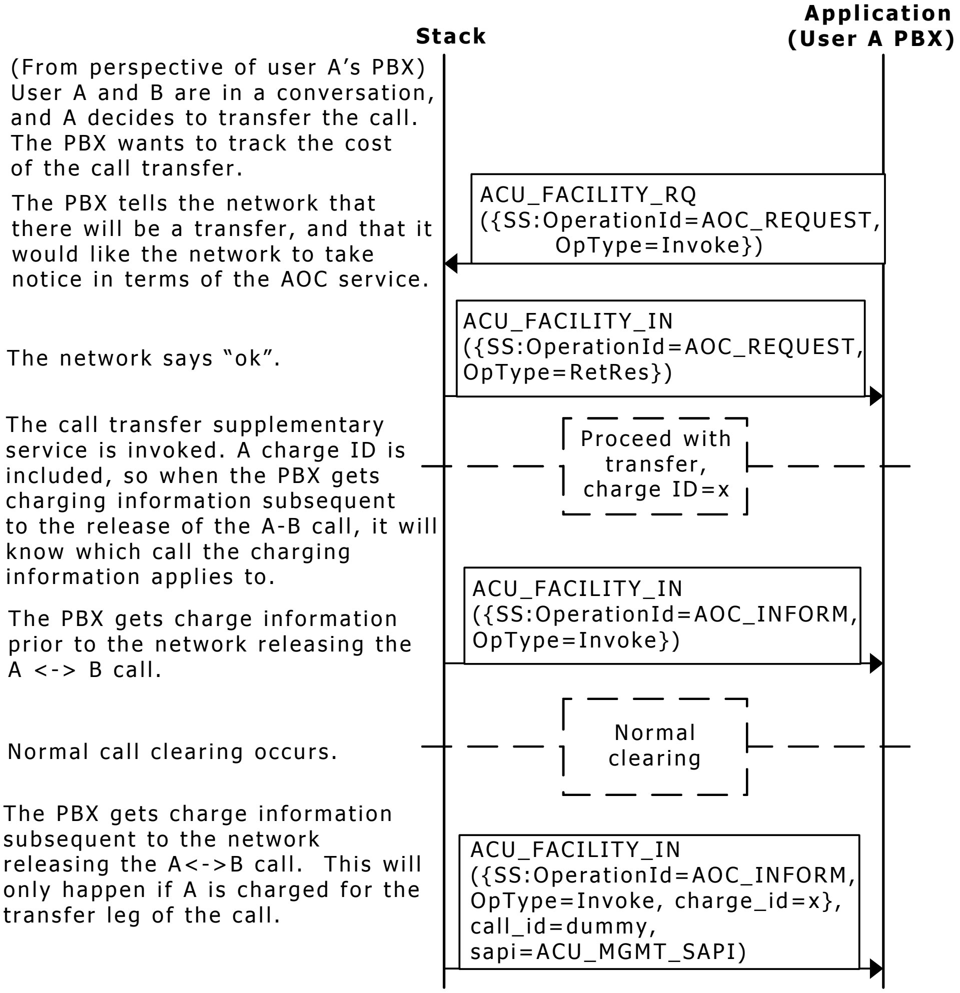
AOC-E and ECT

If the served user continues to be charged for a call after the call is transferred elsewhere, the served user can associate an identifier with the call, so AOC-E information for the call can be returned to the served user even though the served user is no longer involved in the call.

The identifier is specified in the charge_association field in the explicit call transfer or data structure. When AOC-E information is returned at the end of the call, the information is presented to the application using the management SAPI (ACU_MGMT_SAPI) in the AOC_INFORM structure, using a dummy connection ID. The charge_association field in the AcuSSAocEInform data structure is filled in, enabling the application to associate the charging information with the specific invocation of the service.

If an identifier is not specified with the call, no AOC information is returned.

If AOC-E is activated for a call, and ECT is called without a charge identifier included, then the AOC-E service is stopped, and the application receives AOC-E information for the portion of the call prior to the transfer.


Stack
Application (User A PBX)
(From perspective of user A's PBX) User A and B are in a conversation, and A decides to transfer the call. The PBX wants to track the cost of the call transfer.
The PBX tells the network that there will be a transfer, and that it would like the network to take notice in terms of the AOC service.
ACU_FACILITY_RQ ({SS:OperationId=AOC_REQUEST, OpType=Invoke})
The network says "ok".
ACU_FACILITY_IN ({SS:OperationId=AOC_REQUEST, OpType=RetRes})
The call transfer supplementary service is invoked. A charge ID is included, so when the PBX gets charging information subsequent to the release of the A-B call, it will know which call the charging information applies to.
Proceed with transfer, charge ID=x
The PBX gets charge information prior to the network releasing the A <-> B call.
ACU_FACILITY_IN ({SS:OperationId=AOC_INFORM, OpType=Invoke})
Normal call clearing occurs.
Normal clearing
The PBX gets charge information subsequent to the network releasing the A<->B call. This will only happen if A is charged for the transfer leg of the call.
ACU_FACILITY_IN ({SS:OperationId=AOC_INFORM, OpType=Invoke, charge_id=x}, call_id=dummy, sapi=ACU_MGMT_SAPI)

AOC and call transfer

AOC and call deflection services

The AOC service interacts with the call deflection service. This section describes how the services interact.

AOC-S, AOC-D, and call deflection

AOC-S and AOC-D services are not applicable to the user performing the deflection.

AOC-E and call deflection

If the served user continues to be charged for a call after the call is deflected elsewhere, the served user can associate an identifier with the call, so AOC-E information for the call can be returned to the served user even though the served user is no longer involved in the call.

The identifier is specified in the charge_association field in the call deflection data structure. When AOC-E information is returned at the end of the call, the information is presented to the application using the management SAPI (ACU_MGMT_SAPI) in the AOC_INFORM structure, using a dummy connection ID. The charge_association field in the AcuSSAocEInform data structure is filled in, enabling the application to associate the charging information to the specific invocation of the service.

If AOC-E is activated for a call, and the call deflection service is called without a charge identifier included, the AOC-E service is stopped, and the application receives AOC-E information for the portion of the call prior to the deflection.

Configuring the NAI for AOC subscription services

If you have subscribed to advice of charge services, to use the services in this manner you must specify which AOC service you have subscribed to when you start the ISDN stack on an NAI. You can specify this information in the ISDN_PROTOCOL_PARMS_Q931CC structure referenced in the initial call to **isdnStartProtocol**. The fields to fill are as follows:

- aoc_s_presubscribed
- aoc_d_presubscribed
- aoc_e_presubscribed

For more information, see the *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual*.

13 Call identification services

Call identification services overview

This section describes supplementary services related to identification of one party to another. The following identification services are implemented in NMS ISDN:

Identification service	Description
Calling name identification presentation (CNIP)	The called party receives the name of the calling party. Available only under the Q.SIG variant.
Connected name identification presentation (CONP)	The calling party receives the name of the called party. Available only under the Q.SIG variant.
Calling line identification presentation (CLIP)	The called party receives the calling party's address information.
Calling line identification restriction (CLIR)	Prevents the calling party's address information from being presented to called users.
Connected line identification presentation (COLP)	Allows the calling party to determine the connected party's address information.
Connected line identification restriction (COLR)	Restricts the calling party from determining the connected party's address information.

Invoking identification services

None of the identification services require exchange of information using extended data structures. All information is exchanged in fields in the basic call control primitives.

For information on filling the macros and sending the messages documented in this section, see the *Dialogic® NaturalAccess™ ISDN Messaging API Developer's Manual*.

Calling name identification presentation (CNIP)

The CNIP service allows name information of the calling party to be passed to the called party (caller ID).

To send name information, the originating node application populates the CNIP-related fields in an ACU_CONN_RQ primitive. These fields are accessible by means of the following macros:

Macro	Description
Acu_conn_rq_a_calling_name (Acu_conn_rq_a_ss_cnip_name)	Pointer to a buffer containing a calling name.
Acu_conn_rq_calling_name_size (Acu_conn_rq_ss_cnip_name_size)	Size of a buffer containing a calling name.
Acu_conn_rq_calling_name_active (Acu_conn_rq_ss_cnip_name_active)	Indicates CNIP supplementary service is invoked.
Acu_conn_rq_calling_name_pres (Acu_conn_rq_ss_cnip_name_pres)	Calling name presentation mode. Allowed values: ACU_SS_NAME_ALLOWED_ISO8859 ACU_SS_NAME_ALLOWED_T61 ACU_SS_NAME_RESTRICTED_ISO8859 ACU_SS_NAME_RESTRICTED_T61 ACU_SS_NAME_NOT_AVAILABLE

Note: The names of the macros have been modified to make them more descriptive. Both names are currently supported with the deprecated names appearing in parentheses.

Name information is presented to the served user in the ACU_CONN_IN primitive. To receive name information, the served user node checks the value of Acu_conn_in_calling_name_active to determine if the service has been invoked. If it has, the application examines the value of Acu_conn_in_calling_name_pres to determine if the calling name is provided to the served user.

Connected name identification presentation (CONP)

The CONP service provides the calling party with the name information of the called party. The service is available only under Q.SIG.

To send connected name information, the originating node application populates the CONP-related fields in one of the following primitives, depending upon the state of the called party:

Primitive	Scenario	The served user node receives:
ACU_CONN_RS	When the called party answers.	ACU_CONN_CO
ACU_ALERT_RQ	When the called party is being alerted.	ACU_ALERT_IN
ACU_CLEAR_RQ	When the called party is busy.	ACU_CLEAR_IN

The CONP-related fields are accessible by means of the following macros:

Macro	Description
Acu_***xxxx_yy***_a_connected_name (Acu_***xxxx_yy***_a_ss_conp_name)	Pointer to a buffer containing a connected name.
Acu_ ***xxxx_yy***_connected_name_size (Acu_ ***xxxx_yy***_ss_conp_name_size)	Size of a buffer containing a connected name.
Acu_ ***xxxx_yy***_connected_name_active (Acu_ ***xxxx_yy***_ss_conp_name_active)	Indicates that CONP supplementary service is invoked.
Acu_ ***xxxx_yy***_connected_name_pres (Acu_ ***xxxx_yy***_ss_conp_name_pres)	Connected name presentation mode. Allowed values: ACU_SS_NAME_ALLOWED_ISO8859 ACU_SS_NAME_ALLOWED_T61 ACU_SS_NAME_RESTRICTED_ISO8859 ACU_SS_NAME_RESTRICTED_T61 ACU_SS_NAME_NOT_AVAILABLE

Note: The names of the macros have been modified to make them more descriptive. Both names are currently supported with the deprecated names appearing in parentheses.

xxxx_yy is the name of an ACU message primitive minus the ACU_ prefix, in lowercase, such as conn_rq, facility_rq, or notify_in. For example:

Acu_notify_in_a_connected_name

To receive name information, the served user node checks the value of Acu_***xxxx_yy***_connected_name_active to determine if the service has been invoked. If it has, the application examines the value of Acu_***xxxx_yy***_connected_name_pres to determine if the calling name is provided to the served user.

Calling line identification presentation (CLIP)

The CLIP service provides the calling party's address information to the called party. This information consists of:

- The calling party's national (ISDN) number
- The country code, and other international call information (if any)
- (Optional) Subaddress information, if provided by the calling user

This service is available under many variants, including Q.SIG.

CLIP under non-Q.SIG variants

Under most non-Q.SIG variants, the service is generally available to the called user (no subscription is required).

The CLIP service is activated and deactivated by the service provider. When the service is activated, the network automatically invokes the service in the call setup phase. The application need not perform any special activation or invocation operations to use the service.

The calling user can present address information to the network in the ACU_CONN_RQ primitive. If the calling user does not include address information in the primitive, the network fills in the information by default.

To specify address information, the calling user uses the following macros:

Macro	Description
Acu_conn_rq_a_calling_nb	Pointer to buffer containing calling number.
Acu_conn_rq_calling_nb_size	Size of buffer containing calling number.
Acu_conn_rq_a_calling_nb_sub	Pointer to buffer containing calling subaddress.
Acu_conn_rq_calling_nb_sub_size	Size of buffer containing calling subaddress.
Acu_conn_rq_calling_nb_plan	Calling number plan.
Acu_conn_rq_calling_nb_pres	Calling number presentation.
Acu_conn_rq_calling_nb_type	Calling number type.
Acu_conn_rq_calling_nb_sub_odd_even	Called subaddress odd/even.
Acu_conn_rq_calling_nb_sub_type	Calling subaddress number type.

The Acu_conn_rq_calling_nb_pres macro determines if this information is to be made available to the called user. If the calling user sets Acu_conn_rq_calling_nb_pres to N_PRES_ALLOWED, the called user can access this information. If the calling user sets Acu_conn_rq_calling_nb_pres to N_PRES_RESTRICTED, the network invokes the Connected line identification presentation (COLP) service, and blocks this information from the called user.

The called user receives the address information in the ACU_CONN_IN primitive. To access this information, the called user uses the following macros:

Macro	Description
Acu_conn_rq_a_calling_nb	Pointer to buffer containing calling number.
Acu_conn_rq_calling_nb_size	Size of buffer containing calling number.
Acu_conn_rq_a_calling_nb_sub	Pointer to buffer containing calling subaddress.
Acu_conn_rq_calling_nb_sub_size	Size of buffer containing calling subaddress.
Acu_conn_rq_calling_nb_plan	Calling number plan.
Acu_conn_rq_calling_nb_pres	Calling number presentation.
Acu_conn_rq_calling_nb_screen	Calling number screening indicator.
Acu_conn_rq_calling_nb_type	Calling number type.
Acu_conn_rq_calling_nb_sub_odd_even	Called subaddress odd/even.
Acu_conn_rq_calling_nb_sub_type	Calling subaddress number type.

The Acu_conn_in_calling_nb_screen macro indicates the result of the network's screening of the address information. Screening is the process by which the network checks that the address information provided by the calling user is acceptable to the network. Possible values for this macro are:

Value	Meaning
N_SCREEN_USER_PASSED	The number was successfully screened.
N_SCREEN_USER_FAILED	The number was not acceptable to the network.
N_SCREEN_NETWORK_PROVIDED	The number was provided by the network (the calling user did not provide the information directly).
N_SCREEN_USER_PROVIDED	No screening took place.

CLIP under the Q.SIG variant

The Q.SIG application has the role of network. As a provider of the CLIP service, the application is responsible for the following:

- Determining whether the calling party address information is to be presented to the called user, depending upon the presentation setting
- Sending the calling party address information to the called user
- Screening the calling party address information and providing the screening result in the primitive sent to the called user

If the application cannot verify the calling party address information (that is, if the screening result is N_SCREEN_NETWORK_PROVIDED or N_SCREEN_USER_PROVIDED), it can include the default access number for the calling user interface in the message sent to the called user. This information is stored in the following macros:

Macro	Description
Acu_conn_rq_a_calling_nb2	Order to buffer containing second calling number.
Acu_conn_rq_calling_nb2_size	Size of buffer containing second calling number.
Acu_conn_rq_calling_nb2_pres	Presentation of second calling number.
Acu_conn_rq_calling_nb2_screen	Screen indicator of second calling number.
Acu_conn_rq_calling_nb2_type	Type of second calling number.

CLIP and call forwarding services

When a call has been diverted or deflected, and the diverted-to user has been provided with the CLIP service, the diverted-to user receives the address information of the calling user, unless the calling user has subscribed to the CLIR service.

Calling line identification restriction (CLIR)

The CLIR service is used with the CLIP service. The CLIR service blocks calling party address information from being presented to the called user.

This service is available under many variants, including Q.SIG.

The CLIR service can be provided in either permanent or temporary mode.

- Permanent mode: Usually provided to the calling user on a subscription basis. Prevents the calling user's address information from being presented to called users.
- Temporary mode: May be provided to the calling user on a subscription basis or may be generally available. Allows the calling user to decide, on a call-by-call basis, whether to allow the called user access to the calling party address information. When subscribing, the user can specify a default behavior:

Default behavior	Description
Presentation restricted	By default, the network blocks the calling user's address information from the called user.
Presentation not restricted	By default, the network allows the calling user's address information to be presented to the called user.

The calling user can override either default on a call-by-call basis.

If the calling user has subscribed to CLIR in temporary mode, the application determines if the called user can access the address information by setting the Acu_conn_rq_calling_nb_pres macro in the ACU_CONN_RQ message. If the calling user sets Acu_conn_rq_calling_nb_pres to N_PRES_ALLOWED, the called user can access this information. If the calling user sets Acu_conn_rq_calling_nb_pres to N_PRES_RESTRICTED, the information is denied the called user. In this case, calling line information is not included in the ACU_CONN_IN primitive.

Note: A subscription option exists which causes presentation restricted to be overridden.

CLIR and other services

This section describes the interaction between the CLIR service and other supplementary services.

Call diversion, call deflection

If the CLIR service is active, the calling party's address information is not presented to the diverted-to user unless the diverted-to user has the CLIR override subscription option.

Explicit call transfer

The calling user's restriction requirement from the original call is used to restrict the presentation of that user's address information to the transferred-to user.

Connected line identification presentation (COLP)

The COLP service allows the calling party to receive a connected party's address information. The calling party can use this service to obtain the address information of the final connected party (the party causing the connect message transmission at the remote end). This information consists of:

- The calling party's national (ISDN) number
- The country code, and other international call information (if any)
- (Optional) Subaddress information, if provided by the connected party

This service is available under many variants, including Q.SIG.

COLP under non-Q.SIG variants

Under most non-Q.SIG variants, the service is generally available to the calling user (no subscription is required).

The COLP supplementary service is activated and deactivated by the service provider. When the service is activated, the network automatically invokes the service on each outgoing call made by the calling user. The application need not perform any special activation or invocation operations to use the service.

The called user can present address information to the network in the ACU_CONN_RS primitive. If the called user does not include address information in the primitive, the network fills in the information by default.

Macros for specifying address information

To specify address information, the called user uses the following macros:

Macro	Description
Acu_conn_rs_a_connected_nb	Pointer to buffer containing connected number.
Acu_conn_rs_connected_nb_size	Size of buffer containing connected number.
Acu_conn_rs_a_connected_sub	Pointer to buffer containing connected subaddress.
Acu_conn_rs_connected_sub_size	Size of buffer containing connected subaddress.
Acu_conn_rs_connected_nb_pres	Connected number presentation.
Acu_conn_rs_connected_nb_type	Connected number type.
Acu_conn_rs_connected_sub_odd_even	Connected subaddress odd/even.
Acu_conn_rs_connected_sub_type	Connected subaddress type.

The Acu_conn_rs_connected_nb_pres macro determines whether or not this information is to be made available to the calling user. If the calling user sets Acu_conn_rs_connected_nb_pres to N_PRES_ALLOWED, the calling user can access this information. If the called user sets Acu_conn_rs_connected_nb_pres to N_PRES_RESTRICTED, the network invokes the connected line presentation restriction (COLR) service, and blocks this information from the calling user. For more information about COLP, see *Connected line identification restriction (COLR)* on page 116.

Macros for accessing address information in the ACU_CONN_CO primitive

The calling user receives the address information in the ACU_CONN_CO primitive. To access this information, the called user uses the following macros:

Macro	**Description**
Acu_conn_co_a_connected_nb	Pointer to buffer containing connected number.
Acu_conn_co_connected_nb_size	Size of buffer containing connected number.
Acu_conn_co_a_connected_sub	Pointer to buffer containing connected subaddress.
Acu_conn_co_connected_sub_size	Size of buffer containing connected subaddress.
Acu_conn_co_connected_nb_pres	Connected number presentation.
Acu_conn_co_connected_nb_screen	Connected number screening indicator.
Acu_conn_co_connected_nb_type	Connected number type.
Acu_conn_co_connected_sub_odd_even	Connected subaddress odd/even.
Acu_conn_co_connected_sub_type	Connected subaddress type.

Possible values for the screening macro

The Acu_conn_co_connected_nb_screen macro indicates the result of the network's screening of the address information. Screening is the process by which the network checks that the address information provided by the called user is acceptable to the network. Possible values for this macro are:

Value	**Meaning**
N_SCREEN_USER_PASSED	Number successfully screened.
N_SCREEN_USER_FAILED	Number not acceptable to the network.
N_SCREEN_NETWORK_PROVIDED	Number provided by the network (the calling user did not provide the information directly).
N_SCREEN_USER_PROVIDED	No screening took place.

COLP under the Q.SIG variant

The Q.SIG application has the role of network. As a provider of the COLP service, the application is responsible for the following:

- Determining whether the called party address information is to be presented to the calling user depending upon the presentation setting.
- Sending the called party address information to the calling user.
- Screening the called party address information and providing the screening result in the primitive sent to the calling user.

COLP and call forwarding services

If the forwarding user selects the option that the calling user is not notified of call forwarding, the calling user does not receive forwarding notification. In addition, the calling user does not receive COLP information when the call is answered unless the calling user has an override subscription option.

If the forwarding user selects the option that the calling user is notified, but without the forward-to number, the calling user does not receive COLP information when the call is answered unless the calling user has an override subscription option.

Connected line identification restriction (COLR)

The COLR service is used with the COLP service. The COLR service blocks called party address information from being presented to the calling user.

This service is available under many variants, including Q.SIG.

The COLR service can be provided in either permanent or temporary mode:

- Permanent mode: Usually provided to the called user on a subscription basis. Prevents the called user's address information from being presented to calling users.
- Temporary mode: May be provided to the called user on a subscription basis or may be generally available. Allows the called user to decide, on a call-by-call basis, whether to allow the calling user access to the called party address information. When subscribing, the user can specify a default behavior:

Default behavior	Description
Presentation restricted	By default, the network blocks the called user's address information from the calling user.
Presentation not restricted	By default, the network allows the called user's address information to be presented to the calling user.

The called user can override either default on a call-by-call basis.

If the called user has subscribed to COLR in temporary mode, the application determines if the calling user can access the address information by setting the Acu_conn_rs_connected_nb_pres macro in the ACU_CONN_RS message. If the called user sets Acu_conn_rs_connected_nb_pres to N_PRES_ALLOWED, the calling user can access this information. If the called user sets Acu_conn_rs_connected_nb_pres to N_PRES_RESTRICTED, the information is denied the calling user. In this case, called line information is not included in the ACU_CONN_CO primitive.

Note: A subscription option exists that causes presentation restricted to be overridden.

COLR and other services

This section describes the interaction between the COLR service and other supplementary services.

Notify diversion

If a forwarded-to or deflected-to user subscribes to COLR permanent mode, the user's number is not provided with the diversion notification.

If a forwarded-to or deflected-to user subscribes to COLR temporary mode, the user's number is not provided until negotiations with the user have taken place and a positive indication from the user has been received (the default value will not be used). The user's number can still be released on answer after confirmation, or the default can be used.

In either situation, a calling user who subscribes to COLP and has override capability does not receive the forwarded-to or deflected-to user's number as part of the notify diversion message, but can use the override capability to receive COLP information when the call is answered.

Explicit call transfer

The connected user's restriction requirement from the original call is used to restrict the presentation of that user's address information to the transferred-to user.

14 Extended data structure substructures

Overview of extended data structure substructures

The structures in this section are referenced in one or more of the service-specific structures and are defined in *isdnacu.h*.

Each of these structures contains an invoke field, indicating the presence or absence of useful data in the structure. If the field is set to ON, then the stack uses the information in the structure. Otherwise, the information in the structure is ignored. If the successful invocation of the service requires the inclusion of the parameter and the invoke field is OFF, the service request is rejected.

Some structures contain a reference to a string location and string length. The string location is specified using the offset field. This offset value is calculated from the address of the structure containing the offset field.

The len field in a structure contains the length of the data. There is no requirement to null-terminate the string. Note that structures sent from the stack do not include null terminations.

These structures are defined to be 8-byte aligned.

acu_address

Specifies substructures that contain the full address and subaddress of a party.

The invoke fields in the acu_party_num and acu_party_subaddress substructures indicate if the substructures contain information. At least one of these substructures must be invoked.

Structure

```
struct acu_address
{
  uchar                       invoke; /* ON/OFF, indicates presence/  */
  pad7                                /* absence of address           */
  struct acu_party_num        num;    /* Number portion of address    */
  struct acu_party_subaddress sub;    /* Subaddress portion of address*/
};
```

acu_conn_id

Specifies the connection information for a related party.

Structure

```
struct acu_conn_id
{
  uchar  invoke;   /* ON or OFF, indicates presence/absence of conn_id*/
  uchar  board;    /* The board number                                */
  nai_t  nai;      /* The network access identifier                   */
  uchar  conn_num; /* The connection number                           */
  pad4
};
```

acu_party_name

Specifies the name of a party.

The offset field in this structure is calculated from the beginning of this structure (the address of the invoke field).

Structure

```
struct acu_party_name
{
  uchar invoke; /* ON/OFF, indicates presence/absence of party name*/
  uchar pres;   /* From SS_NAME... constants                       */
  uchar offset; /* Offset from start of acu_party_name structure   */
  uchar len;    /* Number of bytes                                 */
  pad4
};
```

acu_party_num

Specifies the presentation indicator, numbering plan, and screen indicator of a party.

The offset field in this structure is calculated from the beginning of this structure (the address of the invoke field).

Structure

```
struct acu_party_num
{    uchar  invoke;                 /* ON/OFF, indicates presence/absence */
                                    /* of this information                */
    uchar  presentation_restricted; /* ON indicates presentation is       */
                                    /* restricted                         */
    uchar number_plan;              /* From SS_NUMBER_PLAN... constants   */
    uchar screen_ind;               /* From SS_SCREEN... constants        */
    uchar offset;                   /* Offset from start of this structure*/
                                    /* (address of invoke field)          */
    uchar len;                      /* Number of bytes                    */
    pad2
};
```

acu_party_subaddress

Specifies the subaddress of a party. This structure is referenced in supplementary service data structures.

The offset field in this structure is calculated from the beginning of this structure (the address of the invoke field).

Structure

```
struct acu_party_subaddress
{
  uchar invoke;   /* ON/OFF, indicates presence/absence of subaddress     */
  uchar pad;
  uchar type;     /* ACU_SUBADDRESS_TYPE_NSAP or ACU_SUBADDRESS_TYPE_USER */
  uchar odd_even_ind; /* ACU_SUBADDRESS_EVEN or ACU_SUBADDRESS_ODD        */
  uchar offset;   /* Offset from start of acu_party_subaddress structure  */
  uchar len;      /* Number of bytes                                      */
  pad2
};
```

op_byte_field

Specifies a uchar value.

Structure

```
struct op_byte_field
{
  uchar invoke; /* ON or OFF, indicates presence/absence of this info*/
  uchar value;  /* Interpretation depends on context                 */
  pad6
};
```

op_long_field

Specifies a ulong value.

Structure

```
struct op_long_field
{
  uchar invoke; /* ON or OFF, indicates presence/absence of this info*/
  pad3
  ulong value;  /* Interpretation depends on context                 */
};
```

op_short_field

Specifies a ushort value.

Structure

```
struct op_short_field
{
  uchar  invoke; /* ON or OFF, indicates presence/absence of this info*/
  pad3
  ushort value;  /* Interpretation depends on context                 */
  pad2
};
```

15 Service-specific extended data structures

Overview of service-specific extended data structures

This section provides an alphabetical list of the extended data structures that are defined in *isdnacu.h*. Each listing includes a list of the fields in the structure (other than the header fields) and when they are used.

acu_ss_act_divert_invoke

Requests activation of call diversion services for a user.

Variant

ETS 300

Operation ID

ACU_OP_ID_ACTIVATE_DIVERSION

OpType

ACU_OP_TYPE_INVOKE

Structure

```
struct acu_ss_act_divert_invoke
{
  struct acu_ext_hdr   ext_hdr;     /* Extension header                  */
  struct acu_ss_hdr    ss_hdr;      /* Supplementary serv. header        */
  struct acu_address   diverted_to; /* New destination of call           */
  struct acu_party_num served_user; /* User number to activate the service*/
                                    /* for. If not invoked,then diversion */
                                    /* applies to entire NAI             */
  struct acu_party_num activating_user; /* User activating the diversion, */
                                    /* if different from the served user  */
  uchar                procedure;   /* From ACU_SS_DIVERSION_ constants   */
  uchar                basic_service;  /* From ACU_SS_BASIC_SERVICE_      */
  pad6                              /* constants                         */
};
```

Fields

Field	Description	Mandatory in messages from ETS 300 user-side application?
diverted_to	New destination of call.	Yes
served_user	User number to activate the service for. If not invoked, then the diversion applies to the entire interface (NAI).	Optional
activating_user	The user activating the diversion, if different from the served user.	Not included
procedure	CFU/CFB/CFNR. From AU_SS_DIVERSION constants.	Yes
basic_service	Voice and data. From AU_SS_BASIC_SERVICE constants. If a basic service other than ACU_SS_BASIC_SERVICE_ALL_SERVICES is specified, call diversion is activated just for calls of that service type.	Yes

acu_ss_act_divert_ret_error

Indicates an error in a call diversion activation request.

Variant

ETS 300

Operation ID

ACU_OP_ID_ACTIVATE_DIVERSION

OpType

ACU_OP_TYPE_RETERR

Structure

```
struct acu_ss_act_divert_ret_error
{
  struct acu_ext_hdr ext_hdr; /* Extension header                 */
  struct acu_ss_hdr  ss_hdr;  /* Supplementary services header    */
  ushort             err_id;  /* From ACT_DIVERT_RETERR_ constants*/
  pad6
};
```

Fields

The err_id field contains the reason for the error, from ACT_DIVERT_RETERR constants.

acu_ss_act_divert_ret_result

Indicates that a call diversion activation request was successful.

Variant

ETS 300

Operation ID

ACU_OP_ID_ACTIVATE_DIVERSION

OpType

ACU_OP_TYPE_RETRES

Structure

```
struct acu_ss_act_divert_ret_result
{
  struct acu_ext_hdr ext_hdr; /* Extension header      */
  struct acu_ss_hdr  ss_hdr;  /* Supp. services header */
};
```

Fields

None.

acu_ss_activate_deflect_invoke

Requests activation of call deflection services for a user.

Variant

ETS 300

Operation ID

ACU_OP_ID_ACTIVATE_DEFLECTION

OpType

ACU_OP_TYPE_INVOKE

Structure

```
struct acu_ss_activate_deflect_invoke
{
  struct acu_ext_hdr ext_hdr;    /* Extension header                */
  struct acu_ss_hdr  ss_hdr;     /* Supp. services header           */
  struct acu_address deflect_to; /* Address to deflect all calls to */
};
```

Fields

The deflect_to field is mandatory. It contains the address to deflect the number to.

acu_ss_activate_deflect_ret_result

Indicates that a call deflection activation request was successful.

Variant

ETS 300

Operation ID

ACU_OP_ID_ACTIVATE_DEFLECTION

OpType

ACU_OP_TYPE_RETRES

Structure

```
struct acu_ss_activate_deflect_ret_result
{
  struct acu_ext_hdr ext_hdr;   /* Extension header      */
  struct acu_ss_hdr  ss_hdr;    /* Supp. services header */
};
```

Fields

None.

acu_ss_aoc_inform_invoke

Indicates incoming advice of charge information. The information is specified in substructures contained in this structure.

Variant

ETS 300

Operation ID

ACU_OP_ID_AOC_INFORM

OpType

ACU_OP_TYPE_INVOKE

Structure

```
struct acu_ss_aoc_inform_invoke
{
  struct acu_ext_hdr ext_hdr; /* Extension header                   */
  struct acu_ss_hdr  ss_hdr;  /* Supp. services header              */
  uchar  aoc_type;            /* From ACU_SS_AOC_TYPE... constants  */
  pad7
  union
  {
    tAcuSSAocSInform aoc_s;   /* aoc_type==AOC_SS_TYPE_AOC_S_INFORM */
    tAcuSSAocDInform aoc_d;   /* aoc_type==AOC_SS_TYPE_AOC_D_INFORM */
    tAcuSSAocEInform aoc_e;   /* aoc_type==AOC_SS_TYPE_AOC_E_INFORM */
  } aoc_data;
};
```

Fields

Field	Description	Mandatory in messages to ETS 300 user-side application?
aoc_type	Type of AOC service returning the data. From AU_SS_AOC_TYPE constants.	Yes
aoc_data	aoc_s contains data if aoc_type= ACU_SS_AOC_TYPE_AOC_S_INFORM aoc_d contains data if aoc_type= ACU_SS_AOC_TYPE_AOC_D_INFORM aoc_e contains data if aoc_type= ACU_SS_AOC_TYPE_AOC_E_INFORM	Yes

acu_ss_aoc_request_invoke

Requests one or more advice of charge services.

Variant

ETS 300

Operation ID

ACU_OP_ID_AOC_REQUEST

OpType

ACU_OP_TYPE_INVOKE

Structure

```
struct acu_ss_aoc_request_invoke
{
  struct acu_ext_hdr ext_hdr; /* Extension header               */
  struct acu_ss_hdr  ss_hdr;  /* Supp. services header          */
  uchar              aoc_s;   /* Request AOC-S services (ON/OFF)*/
  uchar              aoc_d;   /* Request AOC-D services (ON/OFF)*/
  uchar              aoc_e;   /* Request AOC-E services (ON/OFF)*/
  pad5
};
```

Fields

Field	Description	Mandatory in messages from ETS 300 user-side application?
aoc_s	Request activation of AOC-S services (ON/OFF)	Yes
aoc_d	Request activation of AOC-D services (ON/OFF)	Yes
aoc_e	Request activation of AOC-E services (ON/OFF)	Yes

acu_ss_aoc_request_ret_error

Indicates an error in the advice of charge service request. The cause field contains the reason for the error.

Variant

ETS 300

Operation ID

ACU_OP_ID_AOC_REQUEST

OpType

ACU_OP_TYPE_RETERR

Structure

```
struct acu_ss_aoc_request_ret_error
{
  struct acu_ext_hdr ext_hdr; /* Extension header                       */
  struct acu_ss_hdr  ss_hdr;  /* Supplementary services header          */
  ushort             cause;   /* From ACU_SS_AOC_ERR.. constants        */
  uchar              aoc_s;   /* Error applies to AOC-S service (ON/OFF)*/
  uchar              aoc_d;   /* Error applies to AOC-D service (ON/OFF)*/
  uchar              aoc_e;   /* Error applies to AOC-E service (ON/OFF)*/
  pad3
};
```

Fields

Field	Description	Mandatory in messages to ETS 300 user-side application?
cause	Error cause. From AU_SS_AOC_ERR constants.	Yes
aoc_s	(ON/OFF) Error in AOC-S services.	Yes
aoc_d	(ON/OFF) Error in AOC-D services.	Yes
aoc_e	(ON/OFF) Error in AOC-E services.	Yes

acu_ss_aoc_request_ret_result

Indicates that an advice of charge service request was successful.

Variant

ETS 300

Operation ID

ACU_OP_ID_AOC_REQUEST

OpType

ACU_OP_TYPE_INVOKE

Structure

```
struct acu_ss_aoc_request_ret_result
{
  struct acu_ext_hdr ext_hdr; /* Extension header              */
  struct acu_ss_hdr  ss_hdr;  /* Supp. services header         */
  uchar              aoc_s;   /* AOC-S services active (ON/OFF)*/
  uchar              aoc_d;   /* AOC-D services active (ON/OFF)*/
  uchar              aoc_e;   /* AOC-E services active (ON/OFF)*/
  pad5
};
```

Fields

Field	Description	Mandatory in messages to ETS 300 user-side application?
aoc_s	(ON/OFF) AOC-S services active/inactive.	Yes
aoc_d	(ON/OFF) AOC-D services active/inactive.	Yes
aoc_e	(ON/OFF) AOC-E services active/inactive.	Yes

acu_ss_association

Contains a charge ID and charged number. If advice of charge - end of call is activated for a call, and the served user continues to be charged for a call after the call is deflected or transferred elsewhere, the served user can use acu_ss_association to associate an identifier with the call. This allows AOC-E information for the call to be returned to the served user even though the served user is no longer involved in the call.

The invoke field in this structure indicates if this information is included, in cases where this information is optional. If invoke is ON, the information is present. If invoke is OFF, the information is not present.

Structure

```
struct acu_ss_association
{
  uchar                  invoke;     /* ON/OFF, indicates presence/absence*/
  pad7                               /* of this info                      */
  struct op_short_field  charge_id;  /* Unique ID value assigned by app   */
  struct acu_party_num   charged_nb; /* Changed party ID by party_num     */
};
```

Note: If this structure is used, the invoke field in either the charge_id or charge_nb structure must be ON. They should not both be OFF simultaneously.

acu_ss_bridge_calls_invoke

Invokes the bridge calls service for two calls.

Variant

Q.SIG

Operation ID

ACU_OP_ID_BRIDGE_CALLS

OpType

ACU_OP_TYPE_INVOKE

Structure

```
struct acu_ss_bridge_calls_invoke
{
  struct acu_ext_hdr ext_hdr;   /* Extension header                   */
  struct acu_ss_hdr  ss_hdr;    /* Supp. services header              */
  struct acu_conn_id bridge_to; /* Connection ID of call to bridge to */
};
```

Fields

The bridge_to field is mandatory in requests for this service.

acu_ss_bridge_calls_ret_result

Indicates that an explicit invocation of the bridge calls service is successful.

Variant

Q.SIG

Operation ID

ACU_OP_ID_BRIDGE_CALLS

OpType

ACU_OP_TYPE_RETRES

Structure

```
struct acu_ss_bridge_calls_ret_result
{
  struct acu_ext_hdr ext_hdr; /* Extension header      */
  struct acu_ss_hdr  ss_hdr;  /* Supp. services header */
};
```

Fields

None.

acu_ss_deact_divert_invoke

Requests deactivation of call diversion services for a user.

Variant

ETS 300

Operation ID

ACU_OP_ID_DEACTIVATE_DIVERSION

OpType

ACU_OP_TYPE_INVOKE

Structure

```
struct acu_ss_deact_divert_invoke
{
  struct acu_ext_hdr   ext_hdr;        /* Extension header                */
  struct acu_ss_hdr    ss_hdr;         /* Supplementary serv. header      */
  struct acu_party_num served_user;    /* User to deactivate diversion for*/
  struct acu_party_num deactivating_user; /* User initiating the         */
                                       /* deactivation                    */
  uchar                procedure;      /* From ACU_SS_DIVERSION_ constants*/
  uchar                basic_service;  /* From ACU_SS_BASIC_SERVICE_      */
  pad6                                 /* constants                       */
};
```

Fields

Field	Description	Mandatory in messages from ETS 300 user-side app?
served_user	User to deactivate diversion for.	Optional
deactivating_user	User initiating the deactivation.	Not included
procedure	CFU/CFB/CFNR. From AU_SS_DIVERSION constants.	Yes
basic_service	Voice and data. From AU_SS_BASIC_SERVICE constants. If a basic service other than ACU_SS_BASIC_SERVICE_ALL_SERVICES is specified, call diversion is deactivated just for calls of that service type.	Yes

acu_ss_deact_divert_ret_error

Indicates an error in a call diversion deactivation request.

Variant

ETS 300

Operation ID

ACU_OP_ID_DEACTIVATE_DIVERSION

OpType

ACU_OP_TYPE_RETERR

Structure

```
struct acu_ss_act_divert_ret_error
{
  struct acu_ext_hdr ext_hdr; /* Extension header                   */
  struct acu_ss_hdr  ss_hdr;  /* Supp. services header              */
  ushort             err_id;  /* From DEACT_DIVERT_RETERR_ constants*/
  pad6
};
```

Fields

The err_id field contains the reason for the error, from DEACT_DIVERT_RETERR constants.

acu_ss_deact_divert_ret_result

Indicates that a call diversion deactivation request was successful.

Variant

ETS 300

Operation ID

ACU_OP_ID_DEACTIVATE_DIVERSION

OpType

ACU_OP_TYPE_RETRES

Structure

```
struct acu_ss_act_divert_ret_result
{
  struct acu_ext_hdr ext_hdr; /* Extension header      */
  struct acu_ss_hdr  ss_hdr;  /* Supp. services header */
};
```

Fields

None.

acu_ss_deactivate_deflect_invoke

Requests deactivation of call deflection services for a user.

Variant

ETS 300

Operation ID

ACU_OP_ID_DEACTIVATE_DEFLECTION

OpType

ACU_OP_TYPE_INVOKE

Structure

```
struct acu_ss_deactivate_deflect_invoke
{
  struct acu_ext_hdr ext_hdr; /* Extension header      */
  struct acu_ss_hdr  ss_hdr;  /* Supp. services header */
};
```

Fields

None.

acu_ss_deactivate_deflect_ret_result

Indicates that a call deflection deactivation request was successful.

Variant

ETS 300

Operation ID

ACU_OP_ID_DEACTIVATE_DEFLECTION

OpType

ACU_OP_TYPE_RETRES

Structure

```
struct acu_ss_deactivate_deflect_ret_result
{
  struct acu_ext_hdr ext_hdr; /* Extension header      */
  struct acu_ss_hdr  ss_hdr;  /* Supp. services header */
};
```

Fields

None.

acu_ss_deflect_invoke

Invokes call deflection services for a specific call.

Variant

ETS 300

Operation ID

ACU_OP_ID_DEFLECTION

OpType

ACU_OP_TYPE_INVOKE

Structure

```
struct acu_ss_deflect_invoke
{
 struct acu_ext_hdr        ext_hdr;            /*Extension header         */
 struct acu_ss_hdr         ss_hdr;             /*Supp. services header    */
 struct acu_address        deflect_to;         /*Number to direct call to */
 struct acu_ss_association charge_association;/*Optional, used when AOC-E*/
};                                             /*service has been invoked */
```

Fields

Field	**Description**	**Mandatory in messages from ETS 300 user-side app?**
deflect_to	Full address and subaddress of deflected-to party.	Yes
charge_association	Optional charge identifier, used when AOC-E service has been invoked. See *AOC and explicit call transfer (ECT) services* on page 103.	Optional

acu_ss_deflect_ret_error

Indicates an error in a call deflection invocation request.

Variant

ETS 300

Operation ID

ACU_OP_ID_DEFLECTION

OpType

ACU_OP_TYPE_RETERR

Structure

```
struct acu_ss_deflect_ret_error
{
  struct acu_ext_hdr ext_hdr; /* Extension header                 */
  struct acu_ss_hdr  ss_hdr;  /* Supp. services header            */
  ushort             err_id;  /* From DEFLECT_RETERR_... constants */
  pad6
};
```

Fields

The err_id field contains the reason for the error, from DEFLECT_RETERR constants.

acu_ss_deflect_ret_result

Indicates that a call deflection invocation request was successful.

Variant

ETS 300

Operation ID

ACU_OP_ID_DEFLECTION

OpType

ACU_OP_TYPE_RETRES

Structure

```
struct acu_ss_deflect_ret_result
{
  struct acu_ext_hdr ext_hdr; /* Extension header      */
  struct acu_ss_hdr  ss_hdr;  /* Supp. services header */
};
```

Fields

None.

acu_ss_divert_invoke

Requests invocation of call diversion services.

Variant

Q.SIG

Operation ID

ACU_OP_ID_DIVERSION

OpType

ACU_OP_TYPE_INVOKE

Structure

```
struct acu_ss_divert_invoke
{
  struct acu_ext_hdr    ext_hdr;          /* Extension header            */
  struct acu_ss_hdr     ss_hdr;           /* Supp. services header       */
  struct acu_address    divert_to_nb;     /* Number of new called party  */
  struct acu_party_num  divert_from_nb;   /* No. called in last setup, if*/
                                          /*different from orig_called_nb*/
  struct acu_party_num  orig_called_nb;   /* Number called in first setup*/
  struct acu_address    calling_nb;       /* Calling number in last setup*/
  struct acu_party_name calling_name;     /* Calling name in last setup  */
  struct acu_party_name redir_name;       /* Name of user invoking this  */
                                          /* diversion                   */
  struct acu_party_name orig_called_name; /* Name of orig. called party  */
  uchar                 procedure;        /* From ACU_SS_DIVERSION_...   */
                                          /* constants                   */
  uchar                 diversion_count;  /* Passed to app, meaningless  */
                                          /* from application            */
  uchar                 subscription;     /* From ACU_SS_SUBSCRIPTION_   */
  pad5                                    /* constants                   */
};
```

Fields

Field	Description	Mandatory in messages to originating Q.SIG node's app?	Mandatory in messages from served user Q.SIG node's app?
divert_to_nb	New destination of the call.	Yes	Yes
divert_from_nb	Number called in last setup message, if different than orig_called_nb.	Optional	Optional
orig_called_nb	Called number from very first setup message.	Optional	Optional
calling_nb	Calling number from last setup message.	Optional	Optional
calling_name	Calling name from last setup message.	Optional	Optional
redir_name	Name of user invoking this diversion.	Optional	Optional
orig_called_name	Name of original called party.	Optional	Optional
procedure	CFU/CFB/CFNR. From ACU_SS_DIVERSION constants.	Yes	Yes

Field	**Description**	**Mandatory in messages to originating Q.SIG node's app?**	**Mandatory in messages from served user Q.SIG node's app?**
diversion_count	Passed to application for storage, and for use in the NOTIFY_DIVERSION service. This information is meaningless in RQ messages.	Yes	Yes
subscription	From ACU_SS_SUBSCRIPTION constants.	Yes	Yes

acu_ss_divert_ret_error

Indicates an error in a call diversion invocation request.

Variant

Q.SIG

Operation ID

ACU_OP_ID_DIVERSION

OpType

ACU_OP_TYPE_RETERR

Structure

```
struct acu_ss_divert_ret_error
{
  struct acu_ext_hdr ext_hdr; /* Extension header             */
  struct acu_ss_hdr  ss_hdr;  /* Supp. services header        */
  ushort             err_id;  /* From DIVERT_RETERR constants */
  pad6
};
```

Fields

The err_id field contains the reason for the error, from DIVERT_RETERR constants.

acu_ss_divert_ret_result

Indicates that a call diversion invocation request was successful.

Variant

Q.SIG

Operation ID

ACU_OP_ID_DIVERSION

OpType

ACU_OP_TYPE_RETRES

Structure

```
struct acu_ss_divert_ret_result
{
  struct acu_ext_hdr   ext_hdr;           /* Extension header            */
  struct acu_ss_hdr    ss_hdr;            /* Supp. services header       */
  struct op_byte_field reason;            /* Reason for diversion, from  */
                                          /* ACU_SS_DIVERSION_... macros */
  struct op_byte_field basic_service;     /* Applies in CC->APP in some  */
                                          /* variants. From BASIC_SERVICE*/
                                          /* constants                   */
  struct acu_address   calling_party;     /* Calling party               */
  struct acu_address   served_user;       /* Served user                 */
  struct acu_party_num orig_called;       /* Original called number      */
  struct acu_party_num last_redirecting;  /* Last redirecting number     */
};
```

Fields

Field	Description	Mandatory in messages from ETS 300 user-side app?	Mandatory in messages from originating Q.SIG node's app?	Mandatory in messages to diverting user Q.SIG node's app
reason	Reason for diversion. From ACU_SS_DIVERSION constants.	Yes	Not included	Not included
basic_service	Voice, data, etc. From ACU_SS_BASIC_SERVICE constants. If absent, for all served users.	Yes	Not included	Not included
calling_party	Calling party that was diverted. Not used for Q.SIG.	Optional	Not included	Not included
served_user	User the call was diverted for. Not used for Q.SIG.	Optional	Not included	Not included
orig_called	Original called party number. Not used for Q.SIG.	Optional	Not included	Not included
last_redirecting	Last called party number. Not used for Q.SIG.	Optional	Not included	Not included

acu_ss_enquire_divert_invoke

Invokes the enquire diversion service.

Variant

ETS 300

Operation ID

ACU_OP_ID_ENQUIRE_DIVERSION

OpType

ACU_OP_TYPE_INVOKE

Structure

```
struct acu_ss_enquire_divert_invoke
{
  struct acu_ext_hdr   ext_hdr;        /* Extension header                 */
  struct acu_ss_hdr    ss_hdr;         /* Supp. services header            */
  struct acu_party_num served_user; /* Leave uninvoked to get information */
                                       /* on all served numbers            */
  struct acu_party_num enquiring_user; /* For remote Q.SIG                 */
  uchar                procedure;      /* From ACU_SS_DIVERSION_ constants*/
  uchar                basic_service;  /* From SS_BASIC_SERVICE_ constants*/
  pad6
};
```

Fields

Field	Description	Mandatory in messages from ETS 300 user-side app?
served_user	Determines whether detailed information about one user or a list of all users is returned. If a served user is specified in the served_user field, detailed information about the user is returned. If the served_user field is not invoked, the network supplies a list of users	Optional
enquiring_user	Reserved for remote Q.SIG.	Not included
procedure	CFU/CFB/CFNR. From AU_SS_DIVERSION constants.	Yes
basic_service	Voice and data. From AU_SS_BASIC_SERVICE constants. If a basic service is specified, information is returned just for call diversion services activated for that service type.	Yes

acu_ss_enquire_divert_ret_error

Indicates an error in an enquire diversion invocation request.

Variant

ETS 300

Operation ID

ACU_OP_ID_ENQUIRE_DIVERSION

OpType

ACU_OP_TYPE_RETERR

Structure

```
struct acu_ss_enquire_divert_ret_error
{
  struct u4_acu_ext_hdr  ext_hdr; /* extension header */
  struct u4_acu_ss_hdr   ss_hdr;  /* supplementary services header */
  ushort                 err_id;  /* From ENQUIRE_DIVERT_... constants */
  pad6
;
```

Fields

The err_id field contains the cause for the error, from ENQUIRE_DIVERT_RETERR constants.

acu_ss_enquire_divert_ret_result

Indicates that an enquire diversion invocation request was successful.

Variant

ETS 300

Operation ID

ACU_OP_ID_ENQUIRE_DIVERSION

OpType

ACU_OP_TYPE_RETRES

Structure

```
struct acu_ss_enquire_divert_ret_result
{
  struct acu_ext_hdr   ext_hdr;       /* Extension header                     */
  struct acu_ss_hdr    ss_hdr;        /* Supp. services header                */
  struct op_byte_field remote_enabled;/*ON/OFF indicates remotely enabled     */
  struct op_byte_field procedure;     /* From ACU_SS_DIVERSION_ constants     */
  struct op_byte_field basic_service; /* From ACU_SS_BASIC_SERVICE_ constants */
  struct acu_address   diverted_to;   /* Address that served user is          */
                                      /* forwarded to                         */

  struct acu_party_num served_user;   /* Served user's address                */
  uchar                complete;      /* ON when no more messages             */
                                      /* coming, OFF if more coming           */
                                      /* Note: when ON, other fields          */
  pad7                                /* might not be invoked                 */
};
```

Fields

Field	Description	Mandatory in messages from ETS 300 user-side app?
remote_enabled	ON/OFF, indicates remotely enabled.	Not included
procedure	CFU, CFB, CFNR. From AU_SS_DIVERSION constants.	Optional
basic_service	Voice and data. From AU_SS_BASIC_SERVICE constants. If absent, applies to all services.	Optional
diverted_to	Number that the served user is forwarded to. If absent, applies to all served users.	Optional
served_user	Served users address.	Optional
complete	ON when no more messages are coming. OFF if more are coming. **Note:** When ON, other fields might not be invoked.	Yes

acu_ss_notify_diversion_invoke

Performs a notify diversion operation.

Variant

ETS 300/Q.SIG

Operation ID

ACU_OP_ID_NOTIFY_DIVERSION

OpType

ACU_OP_TYPE_INVOKE

Structure

```
struct acu_ss_notify_diversion_invoke
{
  struct acu_ext_hdr    ext_hdr;        /* Extension header              */
  struct acu_ss_hdr     ss_hdr;         /* Supp. services header         */
  struct op_byte_field  reason;         /* Reason for diversion, from    */
                                        /* ACU_SS_DIVERSION_constants    */
  struct op_byte_field  basic_service;  /* From ACU_SS_BASIC_SERVICE_...*/
                                        /* constants                     */
  struct acu_conn_id    related_conn_id;/* Identifies related connection*/
  struct op_byte_field  subscription;   /* Subscription option for       */
                                        /* presentation                  */
  struct acu_party_num  redir_nb;       /* At directed-to user, number   */
                                        /* doing redirection             */
  struct acu_party_num  orig_redir_nb;  /* At directed-to user, first    */
                                        /* number doing redirection      */
  struct acu_party_num  nominated_nb;   /* Indicates new destination     */
                                        /* number to calling user        */
  struct acu_party_name redir_name;     /* Name of user invoking the     */
                                        /* diversion                     */
  struct acu_party_name orig_redir_name;/* At diverted-to user, first    */
                                        /* name doing redirection        */
  uchar                 completed;      /* ON  = diversion completed,    */
                                        /* OFF = diversion in progress   */
  uchar                 diversion_count;/* Passed up to application in   */
  pad6                                  /* acu_ss_divert_invoke          */
};
```

Fields

Field	Description	Mandatory in messages to ETS 300 user-side app?	Mandatory in messages from originating Q.SIG node's app?	Mandatory in messages to originating Q.SIG node's app?	Mandatory in messages to diverted-to user Q.SIG node's app?
reason	Reason for diversion, if available. From AU_SS_DIVERSION constants.	Optional	Yes	Optional	Yes
basic_service	Voice and data. From AU_SS_BASIC_SERVICE constants.	Not included	Not included	Not included	Not included
related_conn_id	Related connection id.	Yes	Yes	Yes	Yes

Field	Description	Mandatory in messages to ETS 300 user-side app?	Mandatory in messages from originating Q.SIG node's app?	Mandatory in messages to originating Q.SIG node's app?	Mandatory in messages to diverted-to user Q.SIG node's app?
subscription	Subscription option for presentation.	Not included	Not included	Optional	Not included
redir_nb	Number of user redirecting call.	Optional	Optional	Not included	Optional
orig_redir_nb	Number of first user redirecting call.	Optional	Optional	Not included	Optional
nominated_nb	ETS 300 only. Indicates new destination number to calling user.	Optional	Not included	Yes	Not included
redir_name	Q.SIG only. Name of user invoking the diversion.	Not included	Optional	Optional	Optional
orig_redir_name	Q.SIG only. Name of original redirecting user.	Not included	Optional	Not included	Optional
completed	ON/OFF. ON indicates diversion is completed. OFF indicates diversion is in progress.	Yes	Not included	Yes	Yes
diversion_count	Passed up to application in acu_ss_divert_invoke structure.	Not included	Yes	Not included	Yes

acu_ss_notify_diversion_ret_result

Indicates that a notify diversion operation was successful.

Variant

ETS 300/Q.SIG

Operation ID

ACU_OP_ID_NOTIFY_DIVERSION

OpType

ACU_OP_TYPE_RETRES

Structure

```
struct acu_ss_notify_diversion_ret_result
{
  struct acu_ext_hdr    ext_hdr;      /* Extension header               */
  struct acu_ss_hdr     ss_hdr;       /* Supp. services header          */
  struct acu_party_name redir_name;   /* Redirection name               */
  ushort                pres_allowed; /* Indicates whether diverted-to  */
  pad6                                /* number can be presented(ON/OFF)*/
};
```

Fields

Field	Description	Mandatory in messages to originating Q.SIG node's app?	Mandatory in messages from diverted-to user Q.SIG node's app?
redir_name	The name of the diverted-to user.	Optional	Optional
pres_allowed	(ON/OFF) Indicates whether the diverted-to user allows presentation of number and name.	Yes	Yes

acu_ss_notify_hold_invoke

Performs a notify hold operation.

Variant

ETS 300/Q.SIG

Operation ID

ACU_OP_ID_NOTIFY_HOLD

OpType

ACU_OP_TYPE_INVOKE

Structure

```
struct acu_ss_notify_hold_invoke
{
  struct acu_ext_hdr   ext_hdr;             /* Extension header         */
  struct acu_ss_hdr    ss_hdr;              /* Supp. services header    */
  struct op_byte_field due_to_alternating; /* ON if alternating between*/
                                          /* connections              */
};
```

The due_to_alternating field is optional, and can be set to ON or OFF. It is set to ON if alternating between connections.

acu_ss_notify_retrieve_invoke

Performs a notify retrieve operation.

Variant

ETS 300/Q.SIG

Operation ID

ACU_OP_ID_NOTIFY_RETRIEVE

OpType

ACU_OP_TYPE_INVOKE

Structure

```
struct acu_ss_notify_retrieve_invoke
{
  struct acu_ext_hdr ext_hdr;                 /* Extension header         */
  struct acu_ss_hdr  ss_hdr;                  /* Supp. services header    */
  struct op_byte_field due_to_alternating;    /* ON if alternating between*/
};                                            /* connections              */
```

The due_to_alternating field is optional. It can be set to ON or OFF. Set to ON if alternating between connections.

acu_ss_notify_tbct_calls_ret_result

Indicates that the transferred call was cleared.

Variant

NI2

Operation ID

ACU_OP_ID_TBCT_CALLS

OpType

ACU_OP_TYPE_RETRES

Structure

```
struct acu_ss_tbct_calls_ret_result
{
 struct u4_acu_ext_hdr ext_hdr;    /* extension header */
 struct u4_acu_ss_hdrss_hdr;       /* supplementary services header */
 DWORD call_tag;                   /* call tag of the transferred call */
 pad4
};
```

Fields

The call_tag field contains the value that identifies the call that was transferred (by TBCT) and has cleared. This field's values range from 1 to 120,000 if the notification to controller feature is available.

Note: The call_tag value is unique for a particular PRI. If the application is served by more that one PRI, the same call_tag can be used for different transferred calls on different PRIs.

acu_ss_notify_transfer_invoke

Performs a notify transfer operation.

Variant

ETS 300/Q.SIG

Operation ID

ACU_OP_ID_NOTIFY_TRANSFER

OpType

ACU_OP_TYPE_INVOKE

Structure

```
struct acu_ss_notify_transfer_invoke
{
  struct acu_ext_hdr    ext_hdr;        /* Extension header             */
  struct acu_ss_hdr     ss_hdr;         /* Supp. services header        */
  struct acu_party_name redir_name;     /* Name of new connected party  */
  struct acu_ss_association charge_id;  /* For AOC-E Interworking, Q.SIG*/
                                        /* only                         */
  struct acu_party_num  redir_nb;       /* Indicates no. of joined party*/
  struct acu_conn_id    joined_conn_id; /* Applies in APPL->CC          */
                                        /* direction                    */
  struct acu_party_subaddress redir_sub;/* Subaddress of joined party   */
  ushort                response_rq;    /* If ON, app must respond      */
  uchar                 call_status;    /* From ACU_SS_CALL_STATUS...   */
                                        /* constants                    */
  uchar                 end_designation;/* Primary/Secondary, Q.SIG only*/
  uchar                 update;         /* If ON, provides additional   */
  pad3                                  /* info for last transfer       */
};
```

Fields

Field	Description	Mandatory in messages to ETS 300 user-side app?	Mandatory in messages from Q.SIG PINX serving user A?	Mandatory in messages to Q.SIG PINX serving users B and C?
redir_name	Indicates supplementary service is invoked, and remaining name fields are valid. For Q.SIG only.	Not included	Optional	Not included
charge_id	Charge identifier. Used when AOC-E service has been invoked. See *AOC and explicit call transfer (ECT) services* on page 103.	Not included	Reserved	Reserved
redir_nb	Indicates number of joined party. This field, joined_conn_id, or redir_sub must be specified in request messages.	Optional	Yes	Optional

Field	Description	Mandatory in messages to ETS 300 user-side app?	Mandatory in messages from Q.SIG PINX serving user A?	Mandatory in messages to Q.SIG PINX serving users B and C?
joined_conn_id	Connection ID for joined party. Must be specified in first invocation of notify transfer for a remote party. Exception: can be left out of request message if only one party is on the network (the other is local). This field, redir_nb, or redir_sub must be specified in subsequent request messages.	Not included	Optional	Not included
redir_sub	Subaddress of joined party. This field, redir_nb, or joined_conn_id must be specified in request messages.	Optional	Optional	Optional
response_rq	ON/OFF. If ON, then the application is requested to respond.	Yes	Not included	Yes
call_status	Joined user alerting or answered. From ACU_SS_CALL_STATUS constants.	Yes	Yes	Yes
end_designation	Primary/secondary. For Q.SIG only. From ACU_SS_END_DESIGNATION constants.	Not included	Yes	Yes
update	If ON, indicates this structure contains updated information not included in previous acu_ss_notify_transfer_invoke structure.	Optional	Optional	Optional

acu_ss_notify_transfer_ret_result

Indicates that a notify transfer operation was successful.

Variant

ETS 300/Q.SIG

Operation ID

ACU_OP_ID_NOTIFY_TRANSFER

OpType

ACU_OP_TYPE_RETRES

Structure

```
struct acu_ss_notify_transfer_ret_result
{
  struct acu_ext_hdr     ext_hdr;    /* Extension header                  */
  struct acu_ss_hdr      ss_hdr;     /* Supp. services header             */
  struct acu_party_name  redir_name; /* Provides name of redirecting party*/
  struct acu_party_num   redir_nb;   /* Provides no. of redirecting party */
  struct acu_party_subaddress redir_sub; /* Provides subaddress of        */
};                                   /* redirecting party                 */
```

Fields

Field	Description	Mandatory in messages from ETS 300 user-side app?	Mandatory in messages from Q.SIG PINX serving user B or C?	Mandatory in messages to Q.SIG PINX serving user A and (B or C)?
redir_name	Indicates supplementary service is invoked and remaining name fields are valid. For Q.SIG only.	Yes	Optional	Not included
redir_nb	Indicates number of joined party.	Not included	Optional	Optional
redir_sub	Indicates subaddress of joined party. If response_rq is on in the invocation request, the application should send this information back.	Not included	Optional	Optional

acu_ss_reject

Indicates that a stack or network error occurred when an attempt was made to invoke or activate a service. This substructure gets returned in an ACU indication or confirmation message.

Variant

All

Operation ID

The operation ID of the supplementary service (for example, ACU_OP_ID_CALL_DEFLECTION)

OpType

ACU_OP_TYPE_REJECT

Structure

```
struct acu_ss_reject
{
  struct acu_ext_hdr ext_hdr;       /* Extension header                 */
  struct acu_ss_hdr  ss_hdr;        /* Supp. services header            */
  uchar              local_cause;   /* From SS_REJECT_LOCAL_ constants  */
  uchar              network_cause; /* From SS_REJECT_NETWORK_ constants*/
  pad6
};
```

Fields

The local_cause and network_cause fields in this structure indicate the cause for the rejection. Refer to ACU_SS_REJECT_LOCAL constants and ACU_SS_REJECT_NETWORK constants for more information.

acu_ss_reminder_diversion_invoke

Requests invocation of remind diversion services.

Variant

ETS 300

Operation ID

ACU_OP_ID_REMIND_DIVERSION

OpType

ACU_OP_TYPE_INVOKE

Structure

```
struct acu_ss_reminder_diversion_invoke
{
  struct acu_ext_hdr ext_hdr; /* Extension header      */
  struct acu_ss_hdr  ss_hdr;  /* Supp. services header */
};
```

Fields

None.

acu_ss_retrieve_invoke

Invokes the call retrieve service.

Variant

ETS 300

Operation ID

ACU_OP_ID_RETRIEVE

OpType

ACU_OP_TYPE_INVOKE

Structure

```
struct acu_ss_retrieve_invoke
{
  struct acu_ext_hdr   ext_hdr;             /* Extension header           */
  struct acu_ss_hdr    ss_hdr;              /* Supp. services header      */
  struct op_byte_field due_to_alternating;  /* ON if alternating between */
};                                          /* connections                */
```

Fields

The due_to_alternating field is optional. It can be set to ON or OFF. Set to ON if alternating between connections.

acu_ss_retrieve_ret_result

Indicates that a call retrieve invocation request was successful.

Variant

ETS 300

Operation ID

ACU_OP_ID_RETRIEVE

OpType

ACU_OP_TYPE_RETRES

Structure

```
struct acu_ss_retrieve_ret_result
{
  struct acu_ext_hdr ext_hdr; /* Extension header      */
  struct acu_ss_hdr  ss_hdr;  /* Supp. services header */
};
```

Fields

None.

16 Advice of Charge substructures

Overview of advice of charge substructures

The substructures in this section are used to return AOC information to a served user.

They are referenced in the AcuSSAocInformInvoke structure returned in an AOC_INFORM operation (or in substructures referenced there).

For more information about these substructures, see *Advice of charge services overview* on page 97.

tAcuSSAocDuration

If advice of charge - start of call (AOC-S) is activated for a call, this structure returns call duration rate information to a served user. This information consists of the currency value for a particular time unit and the length of the time unit.

This structure is referenced in the AcuSSAocSInform structure, which is referenced in the AcuSSAocInformInvoke structure returned in an AOC_INFORM operation.

Structure

```
typedef struct tAcuSSAocDuration
{
  struct op_byte_field multiplier;        /* Applies to currency amount   */
  struct op_long_field currency_amount;   /* No. of currency units charged*/
  struct op_long_field time;              /* Number of time units charged */
  struct op_long_field granularity;       /* Integer or not included      */
  struct op_byte_field time_scale;        /* Applies to time              */
  struct op_byte_field granularity_scale; /* Applies to granularity       */
  struct op_byte_field currency_id_size;  /* 0 if currency ID not included*/
  uchar  currency_id[MAX_CURRENCY_SIZE];  /* Currency ID                  */
  uchar  type_of_charging;                /* From SS_AOC_S_TYPE_OF_CHARGE */
  pad3                                    /* constants                    */
};
```

Fields

Field	Included	Description
multiplier	Mandatory	This field plus currency_amount indicates the currency amount. From ACU_SS_AOC_MULTIPLIER constants.
currency_amount	Mandatory	This field plus multiplier indicates the currency amount.
time	Mandatory	This field plus time_scale indicates the length of time unit.
granularity	Optional	This field plus granularity_scale indicates the time unit applied for calculation of charges by the network. Integer or not included.
time_scale	Mandatory	This field plus time indicates the length of time unit. From ACU_SS_AOC_SCALE constants.
granularity_scale	Optional	This field plus granularity indicates the time unit applied for calculation of charges by the network. Only included if granularity field is included.
currency_id_size	Mandatory	Size of currency_id.
currency_id[MAX_CURRENCY_SIZE]	Mandatory	Currency ID.
type_of_charging	Mandatory	Step function (charge is incurred for the time unit, or part thereof) or continuous (charges are incurred evenly throughout). From ACU_SS_AOC_S_TYPE_OF_CHARGE constants.

tAcuSSAocDInform

Returns advice of charge - during call (AOC-D) information to a served user. It is referenced in the AcuSSAocInformInvoke structure returned in an AOC_INFORM operation.

Structure

```
typedef struct tAcuSSAocDInform
{
  struct op_byte_field currency_id_size; /* Used only if type_of_charging */
                                         /* is CURRENCY_UNITS            */
  struct op_long_field currency_amount;  /* Used only if type_of_charging */
                                         /* is CURRENCY_UNITS            */
  struct op_byte_field multiplier;       /* Used only if type_of_charging */
                                         /* is CURRENCY_UNITS            */
  struct op_byte_field billing_id;       /* From ACU_SS_AOC_BILLING_ID    */
                                         /* constants                    */
  struct acu_ss_association charge_association; /* Returns ID of charge   */
                                         /* for AOC-E after call clearing */
  tRecordedUnits       recorded_units;   /* Used only if type_of_charging */
                                         /* is CHARGING_UNITS            */
  uchar currency_id[MAX_CURRENCY_SIZE];  /* Used only if type_of_charging */
                                         /* is CURRENCY_UNITS            */
  uchar type_of_charging;                /* From SS_AOC_TYPE_OF_CHARGE... */
                                         /* constants                    */
  uchar recorded_charges;                /* From SS_AOC_RECORDED_CHARGES  */
                                         /* constants                    */
  uchar completed;                       /* Set to OFF if additional      */
                                         /* recorded units are queued     */
  pad1                                   /* in a subsequent message       */
};
```

Fields

Field	Description
currency_id_size	Size of currency_id. Used only if type_of_charging is CURRENCY_UNITS.
currency_amount	This value plus multiplier indicates the currency amount. Used only if type_of_charging is CURRENCY_UNITS.
multiplier	This value plus currency_amount indicates the currency amount. From ACU_SS_AOC_MULTIPLIER constants. Used only if type_of_charging is CURRENCY_UNITS.
billing_id	From ACU_SS_AOC_BILLING_ID constants.
charge_association	Not used.
recorded_units	Used only if type_of_charging is CHARGING_UNITS.
currency_id[MAX_CURRENCY_SIZE]	Currency ID. Used only if type_of_charging is CURRENCY_UNITS.
type_of_charging	Step function (charge is incurred for the time unit, or part thereof) or continuous (charges are incurred evenly throughout). From ACU_SS_AOC_TYPE_OF_CHARGE constants.
recorded_charges	From AU_SS_AOC_RECORDED_CHARGES constants.
completed	Set to OFF if additional recorded units are queued in a subsequent message.

tAcuSSAocEInform

Returns advice of charge - end of call (AOC-E) information to a served user. It is referenced in the AcuSSAocInformInvoke structure returned in an AOC_INFORM operation.

Structure

```
typedef struct tAcuSSAocEInform
{
  struct op_byte_field currency_id_size;/* Used only if type_of_charging */
                                        /* is CURRENCY_UNITS             */
  struct op_long_field currency_amount; /* Used only if type_of_charging */
                                        /* is CURRENCY_UNITS             */
  struct op_byte_field multiplier;      /* Used only if type_of_charging */
                                        /* is CURRENCY_UNITS             */
  struct op_byte_field billing_id;      /* From ACU_SS_AOC_BILLING_ID    */
                                        /* constants                     */
  struct acu_ss_association charge_association; /* Returns ID of charge  */
                                        /* for AOC-E after call clearing */
  tRecordedUnits       recorded_units;  /* Used only if type_of_charging */
                                        /* is CHARGING_UNITS             */
  uchar currency_id[MAX_CURRENCY_SIZE]; /* Used only if type_of_charging */
                                        /* is CURRENCY_UNITS             */
  uchar type_of_charging;               /* From SS_AOC_TYPE_OF_CHARGE... */
                                        /* constants                     */
  uchar recorded_charges;               /* From SS_AOC_RECORDED_CHARGES  */
                                        /* constants                     */
  uchar completed;                      /* Set to OFF if additional      */
                                        /* recorded units are queued     */
  pad1                                  /* in a subsequent message       */
};
```

Fields

Field	Description
currency_id_size	Size of currency_id. Used only if type_of_charging is CURRENCY_UNITS.
currency_amount	This field plus multiplier indicates the currency amount. Used only if type_of_charging is CURRENCY_UNITS.
multiplier	This field plus currency_amount indicates the currency amount. From ACU_SS_AOC_MULTIPLIER constants. Used only if type_of_charging is CURRENCY_UNITS.
billing_id	From ACU_SS_AOC_BILLING_ID constants.
charge_association	Not used.
recorded_units	Used only if type_of_charging is CHARGING_UNITS.
currency_id[MAX_CURRENCY_SIZE]	Currency ID. Used only if type_of_charging is CURRENCY_UNITS.
type_of_charging	Step function (charge is incurred for the time unit, or part thereof) or continuous (charges are incurred evenly throughout). From ACU_SS_AOC_TYPE_OF_CHARGE constants.
recorded_charges	From ACU_SS_AOC_RECORDED_CHARGES constants.
completed	Set to OFF if additional recorded units are queued in a subsequent message.

tAcuSSAocSInform

Returns advice of charge - start of call (AOC-S) information to a served user. It is referenced in the AcuSSAocInformInvoke structure returned in an AOC_INFORM operation.

Structure

```
typedef struct tAcuSSAocSInform
{
  uchar type_of_info;          /* From ACU_SS_AOC_TYPE_OF_INFO_ constants*/
  uchar charged_item;          /* Bitmap of charged item constants       */
  uchar charging_rate;         /* One of charging rate constants         */
  pad5
  union
  {
    tAcuSSAocDuration duration; /* charging_rate=CHARGING_RATE_DURATION  */
    tAcuSSAocSpecific specific; /* charging_rate=CHARGING_RATE_SPECIFIC  */
    tAcuSSAocVolume   volume;   /* charging_rate=CHARGING_RATE_VOLUME    */
  }rate;
};
```

Fields

Fields	Included	Description
type_of_info	Mandatory	From ACU_SS_AOC_TYPE_OF_INFO constants. If there is charging rate information returned, this field contains CHARGE_RATE; otherwise, the field contains NOT_AVAILABLE.
charge_item	Optional	Bitmap of charged item constants. From ACU_SS_AOC_CHARGED_ITEM constants.
charging_rate	Optional	The charged item. From ACU_SS_AOC_CHARGING_RATE constants constants. This field contains a value only if the type_of_info field contains CHARGE_RATE.
rate	Optional	References the structure containing data associated with the charging rate.

tAcuSSAocSpecific

If advice of charge - start of call (AOC-S) is activated for a call, this structure returns a specific charge value to the served user. The structure is referenced in the AcuSSAocSInform structure, which is referenced in the AcuSSAocInformInvoke structure returned in an AOC_INFORM operation.

Structure

```
typedef struct tAcuSSAocSpecific
{
  struct op_byte_field multiplier;       /* From ACU_SS_AOC_MULTIPLIER_  */
                                         /* constants                    */
  struct op_long_field currency_amount;  /* No. of currency units charged*/
  struct op_byte_field charge_code;      /* 10                           */
  struct op_byte_field currency_id_size; /* No. of bytes from currency_id*/
  uchar  currency_id[MAX_CURRENCY_SIZE]; /* Currency ID string           */
  uchar  rate_type;                      /* From ACU_SS_AOC_RATE_TYPE_   */
  pad3                                   /* constants                    */
};
```

Fields

Field	Included	Description
multiplier	Mandatory	Yes when rate_type is FLAT_RATE. This value plus currency_amount indicates the currency amount. From ACU_SS_AOC_MULTIPLIER constants.
currency_amount	Mandatory	Yes when rate_type is FLAT_RATE. This value plus multiplier indicates the currency amount. Raw numeric value.
charge_code	Mandatory	Yes when rate_type is SPECIAL_CHARGING. Integer value from 1 to 10. Identifies special charge arrangement for the NAI.
currency_id_size	Mandatory	Number of bytes from currency_id.
currency_id[MAX_CURRENCY_SIZE]	Mandatory	Character array identifying the currency.
rate_type	Mandatory	Free of charge, flat rate (a fixed currency value per event), special charging code, or not available. From ACU_SS_AOC_RATE_TYPE constants.

tAcuSSAocVolume

If advice of charge - start of call (AOC-S) is activated for a call, this structure returns volume rate information to a served user. This information consists of the currency value for a particular volume unit and the type of volume unit.

This structure is referenced in the AcuSSAocSInform structure, which is referenced in the AcuSSAocInformInvoke structure returned in an AOC_INFORM operation.

Structure

```
typedef struct tAcuSSAocVolume
{
  struct op_long_field currency_amount;    /* No of currency units charged*/
  struct op_byte_field currency_id_size;   /* No of bytes from currency_id*/
  struct op_byte_field multiplier;         /* From ACU_SS_AOC_MULTIPLIER_ */
                                           /* constants                  */
  uchar    currency_id[MAX_CURRENCY_SIZE]; /* Currency ID string           */
  uchar    volume_type;                    /* From ACU_SS_AOC_VOLUME_TYPE_*/
  pad3                                     /* constants                  */
};
```

Fields

Field	Included	Description
currency_amount	Mandatory	This value plus multiplier indicates the currency amount. Raw numeric value.
currency_id_size	Mandatory	Number of bytes from currency_id.
multiplier	Mandatory	This value plus currency_amount indicates the currency amount. From ACU_SS_AOC_MULTIPLIER constants.
currency_id[MAX_CURRENCY_SIZE]	Mandatory	Character array identifying the currency.
volume_type	Mandatory	Octet, segment, or message to which UUI has been attached. From ACU_SS_AOC_VOLUME constants.

tRecordedUnits

If advice of charge - during call (AOC-E) or advice of charge - end of call (AOC-E) is activated for a call, this structure returns information to a served user. It is referenced in the AcuSSAocDInform and AcuSSAocEInform structures, which are referenced in the AcuSSAocInformInvoke structure returned in an AOC_INFORM operation.

The invoke field in this structure indicates if this information is included, in cases where this information is optional. If invoke is ON, the information is present. If invoke is OFF, the information is not present.

The num_records field can be used in a loop to examine the arrays of charging unit fields.

Structure

```
typedef struct tRecordedUnits
{
  ulong num_of_charging_units[MAX_RECORDED_UNIT_RECORDS];  /*Number of         */
                                                           /*charging units    */
  uchar type_of_charging_units[MAX_RECORDED_UNIT_RECORDS]; /*Used only if      */
                                                           /*type_of_charging  */
                                                           /*is CHARGING_UNITS */
  uchar invoke;      /* If OFF, then no Recorded Unit info is available        */
  uchar num_records; /* Indicates how many of the array elements are used      */
  pad2
};
```

Fields

Field	Description
num_of_charging_units [MAX_RECORDED_UNIT_RECORDS]	Count of the number of units charged.
type_of_charging_units [MAX_RECORDED_UNIT_RECORDS]	Used only if type_of_charging is CHARGING_UNITS. Can be ACU_SS_AOC_TYPE_OF_CHARGING_UNITS_UNUSED.
invoke	(ON/OFF), indicates presence/absence of this information.
num_records	Indicates how many of the MAX_RECORDED_UNIT_RECORDS are filled in.

17 Constants for isdnacu.h

ACT_DIVERT_RETERR constants

These constants specify the error cause in the err_id field of the acu_ss_act_divert_ret_error extended data structure:

Constant	Description/Notes
ACT_DIVERT_RETERR_NOT_SUBSCRIBED	Served user is not subscribed to the service.
ACT_DIVERT_RETERR_NOT_AVAILABLE	Network cannot allocate the resources at this time.
ACT_DIVERT_RETERR_INVALID_SERVED_NB	Served number is unknown to the network.
ACT_DIVERT_RETERR_INTERAC_NOT_ALLOWED	Interaction with other supplementary service is not allowed.
ACT_DIVERT_RETERR_SERVICE_NOT_PROVIDED	User has not subscribed for the basic service which was applied.
ACT_DIVERT_RETERR_RESOURCE_UNAVAILABLE	Resources cannot be acquired.
ACT_DIVERT_RETERR_INV_DIVERTED_TO_NB	Diverted-to number cannot be routed through the network.
ACT_DIVERT_RETERR_DIV_SPECIAL_SERV_NB	Diversion to fire and emergency is prohibited.
ACT_DIVERT_RETERR_DIV_TO_SERVED_USER_NB	Diversion back to served user number is prohibited.
ACT_DIVERT_RETERR_UNSPECIFIED	Unspecified cause.

ACU_SS_AOC_BILLING_ID constants

These constants specify the billing ID type in the billing_id fields of the following extended data structures:

- tAcuSSAocDInform
- tAcuSSAocEInform

These constants indicate the reason why the served user is being charged:

Constant	Description/Notes
ACU_SS_AOC_BILLING_ID_NORMAL	Normal call setup.
ACU_SS_AOC_BILLING_ID_CREDIT_CARD	Credit card number is being charged.
ACU_SS_AOC_BILLING_ID_REVERSE	Incoming call has reversed charges.
ACU_SS_AOC_BILLING_ID_CFU	Call forwarding, unconditional, has been invoked.
ACU_SS_AOC_BILLING_ID_CFB	Call forwarding, busy, has been invoked.
ACU_SS_AOC_BILLING_ID_CFNR	Call forwarding, no response, has been invoked.
ACU_SS_AOC_BILLING_ID_CD	Call deflection has been invoked.
ACU_SS_AOC_BILLING_ID_TRF	Call transfer has been invoked.

ACU_SS_AOC_CHARGED_ITEM constants

These constants specify the charged item method in the charged_item field of the tAcuSSAocSInform extended data structure.

Note: Network providers only use those charged items that are appropriate to the service provider's charging mechanism. Therefore, in some networks, users may or may not receive some of these items or combinations of items. Different networks may give information about the same call in different ways.

Constant	Description/Notes
ACU_SS_AOC_CHARGED_ITEM_BASIC	(Basic communication charged item) The rate to be applied to the basic communication between the users. If the rate changes, the served user is informed by a repetition of this charged item indicating the new rate.
ACU_SS_AOC_CHARGED_ITEM_ATTEMPT	(Call attempt charged item) Indicates the cost applied for successful call attempt before the called user accepts the call.
ACU_SS_AOC_CHARGED_ITEM_SETUP	(Call setup charged item) The cost applied when the connection between the users is established.
ACU_SS_AOC_CHARGED_ITEM_SPECIAL_CHARGING	(Special charging arrangement charged item) A special arrangement exists for calculating the cost of the call. The use of this charged item is not standardized. Its meaning is a matter for the network operator and the user to which it is sent.
ACU_SS_AOC_CHARGED_ITEM_USER_TO_USER	(User-to-user information transfer charged item) The rate to be applied to the transfer of user-to-user information. If the rate changes, the served user is informed by a repetition of this charged item indicating the new rate.
ACU_SS_AOC_CHARGED_ITEM_SUP_SERV	(Operation of supplementary services charged item) The cost applied for the operation of requested supplementary services.

ACU_SS_AOC_CHARGING_RATE constants

These constants specify the charging rate in the charging_rate field of the tAcuSSAocSInform extended data structure. They are used to indicate which structure to use in the union of AOC-S structures:

Constant	Description/Notes
ACU_SS_AOC_CHARGING_RATE_VOLUME	tAcuSSAocVolume is used in rate union.
ACU_SS_AOC_CHARGING_RATE_SPECIFIC	tAcuSSAocSpecific is used in rate union.
ACU_SS_AOC_CHARGING_RATE_DURATION	tAcuSSAocDuration is used in rate union.

ACU_SS_AOC_ERR constants

These constants specify the error cause in the cause field of the acu_ss_aoc_request_ret_error extended data structure:

Constant	**Description/Notes**
ACU_SS_AOC_ERR_UNSPECIFIED	Error occurred, but the cause is not specified.
ACU_SS_AOC_ERR_NO_CH_INF_AVAIL	Network is unable to acquire the information for this call.

ACU_SS_AOC_MULTIPLIER constants

These constants specify the multiplier in the multiplier field of the following extended data structures:

- tAcuSSAocDuration
- tAcuSSAocDInform
- tAcuSSAocEInform
- tAcuSSAocSpecific
- tAcuSSAocVolume

The multiplier is used to adjust the currency value decimal position. For example, to specify a charge of $17.76, the amount could be set to 1776, the currency ID to $, and the multiplier to AOC_SS_AOC_MULTIPLIER_HUNDREDTH.

Constant	Description/Notes
ACU_SS_AOC_MULTIPLIER_THOUSANDTH	0.001 * amount.
ACU_SS_AOC_MULTIPLIER_HUNDREDTH	0.01 * amount.
ACU_SS_AOC_MULTIPLIER_TENTH	0.1 * amount.
ACU_SS_AOC_MULTIPLIER_ONE	1 * amount.
ACU_SS_AOC_MULTIPLIER_TEN	10 * amount.
ACU_SS_AOC_MULTIPLIER_HUNDRED	100 * amount.
ACU_SS_AOC_MULTIPLIER_THOUSAND	1000 * amount.

ACU_SS_AOC_RATE_TYPE constants

These constants specify the rate type in the rate_type field of the tAcuSSAocSpecific extended data structure:

Constant	**Description/Notes**
ACU_SS_AOC_RATE_TYPE_FREE_OF_CHARGE	No charge incurred for the specified service.
ACU_SS_AOC_RATE_TYPE_FREE_OF_CHARGE_FROM_BEGINNING	No charge incurred for the entire call.
ACU_SS_AOC_RATE_TYPE_FLAT_RATE	Flat rate applies to the billed service.
ACU_SS_AOC_RATE_TYPE_SPECIAL_CHARGING	Special charging arrangement applies to the billed service.
ACU_SS_AOC_RATE_TYPE_NOT_AVAIL	No rate information is currently available for the billed service.

ACU_SS_AOC_RECORDED_CHARGES constants

These constants specify the recorded charges unit type in the recorded_charges fields of the following extended data structures:

- tAcuSSAocDInform
- tAcuSSAocEInform

Constant	Description/Notes
ACU_SS_AOC_RECORDED_CHARGES_CHARGING_UNITS	Charge information contains a metered charging unit count.
ACU_SS_AOC_RECORDED_CHARGES_CURRENCY_UNITS	Charge information contains an absolute currency amount.
ACU_SS_AOC_RECORDED_CHARGES_FREE_OF_CHARGE	No charge is applied to the specified service.

ACU_SS_AOC_S_TYPE_OF_CHARGE constants

These constants specify the charging type in the type_of_charging field of the tAcuSSAocDuration extended data structure:

Constant	Description/Notes
ACU_SS_AOC_S_TYPE_OF_CHARGE_CONTINUOUS	Charge is applied evenly throughout the call.
ACU_SS_AOC_S_TYPE_OF_CHARGE_STEP	Charge is applied at the beginning of the specified time unit.

ACU_SS_AOC_SCALE constants

These constants specify the time scale in the time_scale field of the tAcuSSAocDuration extended data structure.

The scale constant is used with an integer (for example, granularity or time fields) to specify the time unit applied for the calculation of charges by the network.

For example, one way to indicate the network will charge on each 6 second interval is to set the time field to 6 and the time_scale field to ACU_SS_AOC_SCALE_ONE_SEC.

Constant	Description/Notes
ACU_SS_AOC_SCALE_HUNDREDTH_SEC	0.01 seconds
ACU_SS_AOC_SCALE_TENTH_SEC	0.1 seconds
ACU_SS_AOC_SCALE_ONE_SEC	1 second
ACU_SS_AOC_SCALE_TEN_SEC	10 seconds
ACU_SS_AOC_SCALE_ONE_MIN	1 minute
ACU_SS_AOC_SCALE_ONE_HOUR	1 hour
ACU_SS_AOC_SCALE_24_HOURS	24 hours

ACU_SS_AOC_TYPE constants

These constants specify the advice of charge service type in the aoc_type field of the acu_ss_aoc_inform_invoke extended data structure:

Constant	**Description/Notes**
ACU_SS_AOC_TYPE_AOC_S_INFORM	aoc_data union contains a tAcuSSAocSInform structure.
ACU_SS_AOC_TYPE_AOC_E_INFORM	aoc_data union contains a tAcuSSAocEInform structure.
ACU_SS_AOC_TYPE_AOC_D_INFORM	aoc_data union contains a tAcuSSAocDInform structure.

ACU_SS_AOC_TYPE_OF_CHARGE constants

These constants specify the type of charge in the type_of_charging fields of the following extended data structures:

- tAcuSSAocDInform
- tAcuSSAocEInform

Constant	Description/Notes
ACU_SS_AOC_TYPE_OF_CHARGE_SUBTOTAL	Charge is only a subtotal of the charges to be attributed to the call.
ACU_SS_AOC_TYPE_OF_CHARGE_TOTAL	Charge is a total of the charges to be attributed to the call.
ACU_SS_AOC_TYPE_OF_CHARGE_NOT_AVAILABLE	No information is available.

ACU_SS_AOC_TYPE_OF_INFO constants

These constants specify the information type in the type_of_info field of the tAcuSSAocSInform extended data structure:

Constant	**Description/Notes**
ACU_SS_AOC_TYPE_OF_INFO_CHARGE_RATE	Charge rate information is included.
ACU_SS_AOC_TYPE_OF_INFO_NOT_AVAILABLE	No further information is available.

ACU_SS_AOC_VOLUME constants

These constants specify the volume type in the volume_type field of the tAcuSSAocVolume extended data structure:

Constant	Description/Notes
ACU_SS_AOC_VOLUME_TYPE_OCTET	User-to-user charged on an octet basis.
ACU_SS_AOC_VOLUME_TYPE_SEGMENT	User-to-user charged on a segment basis.
ACU_SS_AOC_VOLUME_TYPE_MESSAGE	User-to-user charged on a per-message basis.

ACU_SS_BASIC_SERVICE constants

These constants specify the basic service type in the basic_service fields of the following extended data structures:

- acu_ss_notify_diversion_invoke
- acu_ss_divert_ret_result
- acu_ss_act_divert_invoke
- acu_ss_deact_divert_invoke
- acu_ss_enquire_divert_invoke
- acu_ss_enquire_divert_ret_result

Constant	Description/Notes
ACU_SS_BASIC_SERVICE_ALL_SERVICES	All services.
ACU_SS_BASIC_SERVICE_SPEECH	Speech.
ACU_SS_BASIC_SERVICE_UNRESTR_DIG_INFO	Unrestricted digital information.
ACU_SS_BASIC_SERVICE_AUDIO_3K1HZ	Audio 3k1Hz.
ACU_SS_BASIC_SERVICE_UNR_D_INFO_TONE_AN	Unrestricted digital information with tones and announcements.
ACU_SS_BASIC_SERVICE_TELEPHONY_3K1HZ	Telephony 3k1Hz.
ACU_SS_BASIC_SERVICE_TELETEX	Teletex.
ACU_SS_BASIC_SERVICE_TELEFAX_GROUP4_CL1	Telefax Group4 class1.
ACU_SS_BASIC_SERVICE_VIDEOTEX_SYNT_BASE	Videotex syntax based.
ACU_SS_BASIC_SERVICE_VIDEO_TELEPHONY	Video telephony.
ACU_SS_BASIC_SERVICE_TELEFAX_GROUP2_3	Telefax Group2-3.
ACU_SS_BASIC_SERVICE_TELEPHONY_7KHZ	Telephony 7kHz.

ACU_SS_CALL_STATUS constants

These constants specify the status of the call in the call_status fields in the following extended data structure:

- acu_ss_notify_transfer_invoke

Constant	Description/Notes
ACU_SS_CALL_STATUS_NOT_SPECIFIED	Information is not available.
ACU_SS_CALL_STATUS_ALERTING	Connection has not progressed to the active state.
ACU_SS_CALL_STATUS_ANSWERED	Connection has progressed to the active state.

ACU_SS_DIVERSION constants

These constants specify the reason for diversion in the reason fields of the following extended data structures:

- acu_ss_notify_diversion_invoke
- acu_ss_divert_ret_result

These constants specify the diversion procedure in the procedure fields of the following extended data structures:

- acu_ss_act_divert_invoke
- acu_ss_deact_divert_invoke
- acu_ss_divert_invoke
- acu_ss_enquire_divert_invoke
- acu_ss_enquire_divert_ret_result

Constant	Description/Notes
ACU_SS_DIVERSION_UNKNOWN	Call was diverted for unknown reasons.
ACU_SS_DIVERSION_CFU	Network immediately diverted the call.
ACU_SS_DIVERSION_CFB	Network diverted the call after it determined the user was busy.
ACU_SS_DIVERSION_CFNR	Network diverted the call after the served user did not answer.
ACU_SS_DIVERSION_CD_ALERTING	Served user has deflected the call after the alerting state was entered.
ACU_SS_DIVERSION_CD_IMMEDIATE	Served user has deflected the call immediately.
ACU_SS_DIVERSION_CD	Used when an unknown call deflection cause occurs.

ACU_SS_END_DESIGNATION constants

These constants specify the end designation in the end_designation fields in the acu_ss_notify_transfer_invoke extended data structure:

Constant	Description/Notes
ACU_SS_END_DESIGNATION_NOT_SPECIFIED	For use in some variants (ETS 300).
ACU_SS_END_DESIGNATION_PRIMARY	Party associated with the connection is the primary party in the new connection. Only valid if the connection is in the active state.
ACU_SS_END_DESIGNATION_SECONDARY	Party associated with the connection is the secondary party in the new connection.

ACU_SS_NAME constants

These constants specify the name presentation in the pres field of the acu_party_name extended data structure:

Constant	**Description/Notes**
ACU_SS_NAME_ALLOWED_ISO8859	Name can be presented to the user and is encoded in ISO-8859 format.
ACU_SS_NAME_ALLOWED_T61	Name can be presented to the user and is encoded in T-61 format.
ACU_SS_NAME_RESTRICTED_ISO8859	Name cannot be presented to the user and is encoded in ISO-8859 format.
ACU_SS_NAME_RESTRICTED_T61	Name cannot be presented to the user and is encoded in T-61 format.
ACU_SS_NAME_NOT_AVAILABLE	Name is not available.

ACU_SS_NUMBER_PLAN constants

These constants specify the numbering plan in the number_plan field of the acu_party_num extended data structure.

This set of constants is a combination of the type of number and numbering plan identification fields included in the underlying protocol party number information elements. If the application wishes to access these fields directly, or if no macro corresponds to the desired/received value, then the following formula can be used:

ACU_SS_NUMBER_PLAN_X_Y = (***t_of_num*** << 4) + ***num_plan_id***

where the ***t_of_num*** and ***num_plan_id*** fields are defined in the relevant specifications (such as Q.931).

Constant	**Description/Notes**
ACU_SS_NUMBER_PLAN_UNKNOWN	No information about the number.
ACU_SS_NUMBER_PLAN_PUBLIC_UNKNOWN	Public number plan.
ACU_SS_NUMBER_PLAN_PUBLIC_INTERNATIONAL	Public number plan, international number.
ACU_SS_NUMBER_PLAN_PUBLIC_NATIONAL	Public number plan, national number.
ACU_SS_NUMBER_PLAN_PUBLIC_NETWORK_SPECIFIC	Public number plan, network specific number.
ACU_SS_NUMBER_PLAN_PUBLIC_SUBSCRIBER	Public number plan, subscriber number.
ACU_SS_NUMBER_PLAN_PUBLIC_ABBREVIATED	Public number plan, abbreviated number.
ACU_SS_NUMBER_PLAN_PRIVATE_UNKNOWN	Private number plan.
ACU_SS_NUMBER_PLAN_PRIVATE_LEVEL_1_REGIONAL	Private number plan, level 1 region formatted number.
ACU_SS_NUMBER_PLAN_PRIVATE_LEVEL_2_REGIONAL	Private number plan, level 2 region formatted number.
ACU_SS_NUMBER_PLAN_PRIVATE_PTN_SPECIFIC	Private number plan, PTN specific formatted number.
ACU_SS_NUMBER_PLAN_PRIVATE_LOCAL	Private number plan, local formatted number.

ACU_SS_REJECT_LOCAL constants

These constants specify the reason for rejection in the local_cause field of the acu_ss_reject extended data structure:

Constant	Description/Notes
ACU_SS_REJECT_LOCAL_UNUSED_FIELD	Field does not apply (reject originated from network).
ACU_SS_REJECT_LOCAL_UNSPECIFIED	Unknown reject cause.
ACU_SS_REJECT_LOCAL_INVALID_STATE	Local stack is not in a state to accept the service request.
ACU_SS_REJECT_LOCAL_NOT_IMPLEMENTED	Local stack has not implemented the service for this variant.
ACU_SS_REJECT_LOCAL_NO_RESPONSE	Network has not responded to the service request.
ACU_SS_REJECT_LOCAL_INVALID_CHAN	Channel specified is not acceptable.
ACU_SS_REJECT_LOCAL_INVALID_CONN_ID	Associated connection is not acceptable.
ACU_SS_REJECT_LOCAL_TEMP_BUSY	Service cannot be provided due to current allocations of the necessary resources.
ACU_SS_REJECT_LOCAL_MISSING_PARAMETER	Necessary information was not provided.

ACU_SS_REJECT_NETWORK constants

These constants specify the reason for rejection in the network_cause field of the acu_ss_reject extended data structure:

Constant	Description/Notes
ACU_SS_REJECT_NETWORK_UNUSED_FIELD	Local stack rejected the service request.
ACU_SS_REJECT_NETWORK_UNSPECIFIED	Cause is not known.
ACU_SS_REJECT_NETWORK_NOT_SUBSCRIBED	Served user is not subscribed to the service invoked.
ACU_SS_REJECT_NETWORK_NOT_IMPLEMENTED	Network has not implemented the requested service.
ACU_SS_REJECT_NETWORK_INVALID_STATE	Network cannot process the request in the current connection state.
ACU_SS_REJECT_NETWORK_BEARER_CAP_ERROR	There is an incompatibility in the bearer channel.
ACU_SS_REJECT_NETWORK_CHAN_NOT_AVAIL	Channel is not available.
ACU_SS_REJECT_NETWORK_REQUESTED_CHAN_NOT_AVAIL	Requested channel is not available.
ACU_SS_REJECT_NETWORK_DUPLICATE_INVOKE	Switch received more than one invoke component in a facility IE, or the switch received the same invoke identifier included in multiple invoke components.
ACU_SS_REJECT_NETWORK_UNRECOGNIZED_OP	Switch received a facility IE that includes an invoke component with an unrecognized operation.
ACU_SS_REJECT_NETWORK_RESOURCE_LIMITATION	Number of transfer requests per 10 seconds or the number of active transfers exceeds the provisioned maximum. The transfer operation fails.

ACU_SS_SCREEN constants

These constants specify the screen_ind field of the acu_party_num extended data structure:

Constant	Description/Notes
ACU_SS_SCREEN_UNUSED_FIELD	Field is not used or applicable.
ACU_SS_SCREEN_USER_PROV_NOT_SCREENED	User provided the party information and it has not been verified.
ACU_SS_SCREEN_USER_PROV_VERIFIED	User provided the party information and it has been validated by the network.
ACU_SS_SCREEN_USER_PROV_FAILED_VERIFICATION	User provided the party information and the network verification failed.
ACU_SS_SCREEN_NETWORK_PROV	Network provided the party information.

ACU_SS_SUBSCRIPTION constants

These constants specify the subscription in the subscription field of the acu_ss_divert_invoke extended data structure:

Constant	**Description/Notes**
ACU_SS_SUBSCRIPTION_NO_NOTIFY	Diverted party should not be notified of the diversion.
ACU_SS_SUBSCRIPTION_NOTIFY_WITHOUT_NB	Diverted party should not be given the new number, but can be notified of the diversion.
ACU_SS_SUBSCRIPTION_NOTIFY_WITH_NB	All diversion information can be shared with the diverted party.

DEACT_DIVERT_RETERR constants

These constants specify the error cause in the err_id field of the acu_ss_deact_divert_ret_error extended data structure:

Constant	Description/Notes
DEACT_DIVERT_RETERR_NOT_SUBSCRIBED	Served user is not subscribed to this service.
DEACT_DIVERT_RETERR_NOT_AVAILABLE	Supplementary service is not available for the indicated basic service.
DEACT_DIVERT_RETERR_INVALID_SERVED_NB	Served user number could not be found.
DEACT_DIVERT_RETERR_NOT_ACTIVATED	Activation step never completed.
DEACT_DIVERT_RETERR_UNSPECIFIED	Unspecified cause.

DEFLECT_RETERR constants

These constants specify the error cause in the err_id field of the acu_ss_deflect_ret_error extended data structure:

Constant	Description/Notes
DEFLECT_RETERR_NOT_SUBSCRIBED	Served user is not subscribed to this service.
DEFLECT_RETERR_NOT_AVAILABLE	Supplementary service is not available for the indicated basic service.
DEFLECT_RETERR_SS_INTERACTION_NOT_ALLOWED	Interaction with other supplementary service not allowed.
DEFLECT_RETERR_INVALID_DIVERTED_TO_NB	Diverted-to number could not be routed through the network.
DEFLECT_RETERR_SPECIAL_SERVICE_NB	Deflection to fire and emergency is prohibited.
DEFLECT_RETERR_DIVERSION_TO_SERVED_USER_NB	Deflection back to served user number is prohibited.
DEFLECT_RETERR_NUMBER_OF_DIVERSIONS_EXCEEDED	Network has reached the maximum number of diversions for a call.
DEFLECT_RETERR_INCOMING_CALL_ACCEPTED	Basic call control signals have overruled the invocation of the deflect service.
DEFLECT_RETERR_REQUEST_ALREADY_ACCEPTED	Call has progressed too far in the basic call state model.
DEFLECT_RETERR_UNSPECIFIED	Unspecified cause.

DIVERT_RETERR constants

These constants specify the error cause in the err_id field of the acu_ss_divert_ret_error extended data structure:

Constant	Description/Notes
DIVERT_RETERR_NOT_SUBSCRIBED	Served user is not subscribed to the service.
DIVERT_RETERR_NOT_AVAILABLE	Basic service is not available.
DIVERT_RETERR_RESOURCE_UNAVAILABLE	No resources available to invoke the service.
DIVERT_RETERR_INVALID_DIVERTED_TO_NB	Diverted-to number could not be routed.
DIVERT_RETERR_SPECIAL_SERVICE_NB	Diversion to special number (for example, fire) is prohibited.
DIVERT_RETERR_DIVERSION_TO_SERVED_USER_NB	Diversion to served user is prohibited.
DIVERT_RETERR_NUMBER_OF_DIVERSIONS_EXCEEDED	Call has reached the maximum number of diversions.
DIVERT_RETERR_SS_INTERACTION_NOT_ALLOWED	Interaction with other supplementary service not allowed.
DIVERT_RETERR_UNSPECIFIED	Unspecified cause.
DIVERT_RETERR_NO_RESPONSE	Diverted-to party failed to respond to the call.
DIVERT_RETERR_DIVERTED_LEG_FAILED	Diverted-to party failed to answer the call.

ENQUIRE_DIVERT_RETERR constants

These constants specify the error cause in the err_id field of the acu_ss_enquire_divert_ret_error extended data structure:

Constant	Description/Notes
ENQUIRE_DIVERT_RETERR_NOT_SUBSCRIBED	Served user is not subscribed to the service.
ENQUIRE_DIVERT_RETERR_NOT_AVAILABLE	Information is not available.
ENQUIRE_DIVERT_RETERR_INVALID_SERVED_NB	Served user number is not recognized.
ENQUIRE_DIVERT_RETERR_UNSPECIFIED	Unspecified cause.

Index