



CentreVu[®] Computer-Telephony

Release 10.1, Version 1

TSAPI Version 2

Private Data Version 6

Programmer's Guide

for

DEFINITY[®]

Enterprise Communications Server

Issue 1
December 2001

Copyright © 2001 Avaya, Inc.
All Rights Reserved
Printed in USA

Notice

Every effort was made to ensure that the information in this book was complete and accurate at the time of printing. However, information is subject to change.

Preventing Toll Fraud

"Toll fraud" is the unauthorized use of your telecommunications system by an unauthorized party (for example, a person who is not a corporate employee, agent, subcontractor, or working on your company's behalf). Be aware that there may be a risk of toll fraud associated with your system and that, if toll fraud occurs, it can result in substantial additional charges for your telecommunications services.

Avaya Fraud Intervention

If you suspect you are being victimized by toll fraud and you need technical support or assistance, call the appropriate BCS National Customer Care Center telephone number. Users of the MERLIN®, PARTNER®, and System 25 products should call 1 800 628-2888. Users of the System 75, System 85, DEFINITY® Generic 1, 2 and 3, and DEFINITY® ECS products should call 1 800 643-2353.

Providing Telecommunications Security

Telecommunications security (of voice, data, and/or video communications) is the prevention of any type of intrusion to (that is, either unauthorized or malicious access to or use of your company's telecommunications equipment) by some party.

Your company's "telecommunications equipment" includes both this Avaya product and any other voice/data/video equipment that could be accessed via this Avaya product (that is, "networked equipment").

An "outside party" is anyone who is not a corporate employee, agent, subcontractor, or working on your company's behalf.

Whereas, a "malicious party" is anyone (including someone who may be otherwise authorized) who accesses your telecommunications equipment with either malicious or mischievous intent.

Such intrusions may be either to/through synchronous (time-multiplexed and/or circuit-based) or asynchronous (character-, message-, or packet-based) equipment or interfaces for reasons of:

- Utilization (of capabilities special to the accessed equipment)
- Theft (such as, of intellectual property, financial assets, or toll-facility access)
- Eavesdropping (privacy invasions to humans)
- Mischief (troubling, but apparently innocuous, tampering)
- Harm (such as harmful tampering, data loss or alteration, regardless of motive or intent)

Be aware that there may be a risk of unauthorized intrusions associated with your system and/or its networked equipment. Also realize that, if such an intrusion should occur, it could result in a variety of losses to your company (including, but not limited to, human/data privacy, intellectual property, material assets, financial resources, labor costs, and/or legal costs).

Your Responsibility for Your Company's Telecommunications Security

The final responsibility for securing both this system and its networked equipment rests with you – an Avaya customer's system administrator, your telecommunications peers, and your managers. Base the fulfillment of your responsibility on acquired knowledge and resources from a variety of sources including but not limited to:

- Installation documents
- System administration documents
- Security documents
- Hardware-/software-based security tools
- Shared information between you and your peers
- Telecommunications security experts

To prevent intrusions to your telecommunications equipment, you and your peers should carefully program and configure your:

- Avaya provided telecommunications systems and their interfaces
- Avaya provided software applications, as well as their underlying hardware/software platforms and interfaces
- Any other equipment networked to your Avaya products

Avaya does not warrant that this product or any of its networked equipment is either immune from or will prevent either unauthorized or malicious intrusions. Avaya will not be responsible for any charges, losses, or damages that result from such intrusions.

Trademarks

Adobe, Adobe Acrobat, and the Adobe logo are registered trademarks of Adobe Systems, Inc.

CallVisor, CentreVu, DEFINITY, and the Avaya logotype are registered trademarks of Avaya, Inc.

DEFINITY ONE, and DEFINTY PROLOGIX are trademarks of Avaya, Inc.

Novell and NetWare are registered trademarks of Novell, Inc.

Microsoft, Windows, Windows NT, and the Microsoft logo are registered trademarks and Windows 95 is a trademark of Microsoft.

All products and company names are trademarks or registered trademarks of their respective holders.

Obtaining Products

To learn more about Avaya products and to order products, contact Avaya at: **1 800 451 2100**

Warranty

Avaya provides a limited warranty on this product. Refer to the "Avaya Network Software License Agreement" provided with your package.

Comments

If you have comments, complete and return the comment card at the end of this document.

Contents

1	Introduction	1-1
	■ Purpose and Scope	1-1
	■ Intended Audience	1-2
	■ Terminology	1-2
	■ Conventions	1-4
	■ Related Documents	1-5
	CSTA Services Documents	1-5
	■ New Features for Private Data Version 6	1-6
	■ Customer Support	1-6
	■ How to Comment on This Document	1-6

2	TSAPI Architecture Overview	2-1
	■ Purpose of CSTA	2-1
	■ CSTA Standard and API Specification	2-1
	■ CSTA Communication Layers Model	2-2
	CSTA Client/Server Operational Model	2-3
	■ CSTA Client/Server Session and Operation Invocation Model	2-4
	■ Call Control and Call Events	2-4
	■ G3 CSTA System Overview	2-4
	■ G3 CSTA Software Overview	2-5
	DEFINITY Generic 3 PBX Driver (G3PD)	2-6
	■ G3 Private Data Library Overview	2-7
	■ TSAPI Software Components Overview	2-8
	TSAPI Library	2-8
	Telephony Server (Tserver) EXE/NLM	2-8
	Multiple CTI Links to a G3PD	2-9
	■ Progress and Status of the CTI Link on NetWare	2-11
	■ Progress and Status of the CTI Link on Windows NT	2-13

Contents

3	G3 CSTA Service Groups	3-1
	■ Supported Services and Service Groups	3-1
	■ Transferring or Conferencing a Call Together with Screen Pop Information	3-7
	CSTA Services Used to Conference or Transfer Calls	3-8
	Using Original Call Information to Pop a Screen	3-9
	Using UUI to Pass Information to Remote Applications	3-10
	■ TSAPI Version Control	3-11
	■ Private Data Version Control	3-12
	Private Data Version Feature Support	3-13
	CSTAGetAPICaps Confirmation Interface Structures for Private Data Versions 4, 5, and 6	3-14
	Private Data Feature Summary	3-15
	■ Migration from Private Data Version 5 to Private Data Version 6	3-19
	Private Data Function Changes	3-20
	Set Agent State	3-21
	Private Data Sample Code	3-22
	Sample Code 1	3-22
	Sample Code 2	3-24
	Sample Code 3	3-26
	■ G3 CSTA Objects	3-28
	CSTA Object: Device	3-28
	Device Type	3-28
	Device Class	3-29
	Device Identifier	3-29
	Device Identifier Syntax	3-31
	CSTA Device ID Type (Private Data Version 4 and Earlier)	3-32
	CSTA Device ID Type (with Private Data Version 5 and Later)	3-32
	CSTA Object: Call	3-35
	Call Identifier (callID)	3-35
	Call Identifier Syntax	3-35
	Call State	3-35
	CSTA Object: Connection	3-35

Contents

Connection Identifier (connectionID)	3-36
Connection Identifier Conflict	3-36
Connection Identifier Syntax	3-36
Connection State	3-37
Connection State Syntax	3-39
■ G3 CSTA System Capacity	3-40
■ Multiple Telephony Server Considerations	3-43
■ Multiple CTI Link Considerations	3-43
■ Format and Conventions	3-45
■ Common ACS Parameter Syntax	3-48
■ CSTAUniversalFailureConfEvent	3-49
■ ACSUniversalFailureConfEvent	3-55

4	Call Control Service Group	4-1
■	Overview	4-1
	Alternate Call Service	4-2
	Answer Call Service	4-2
	Clear Call Service	4-3
	Clear Connection Service	4-3
	Conference Call Service	4-3
	Consultation Call Service	4-4
	Consultation Direct-Agent Call Service	4-4
	Consultation Supervisor-Assist Call Service	4-5
	Deflect Call Service	4-5
	Hold Call Service	4-5
	Make Call Service	4-6
	Make Direct-Agent Call Service	4-6
	Make Predictive Call Service	4-7
	Make Supervisor-Assist Call Service	4-7
	Pickup Call Service	4-7
	Reconnect Call Service	4-8
	Retrieve Call Service	4-8
	Transfer Call Service	4-8
■	Alternate Call Service	4-9

Issue 1 — December 2001

Contents

■ Answer Call Service	4-13
■ Clear Call Service	4-18
■ Clear Connection Service	4-20
■ Conference Call Service	4-27
■ Consultation Call Service	4-33
■ Consultation Direct-Agent Call Service	4-42
■ Consultation Supervisor-Assist Call Service	4-51
■ Deflect Call Service	4-60
■ Hold Call Service	4-65
■ Make Call Service	4-69
■ Make Direct-Agent Call Service	4-82
■ Make Predictive Call Service	4-91
■ Make Supervisor-Assist Call Service	4-103
■ Pickup Call Service	4-113
■ Reconnect Call Service	4-118
■ Retrieve Call Service	4-126
■ Send DTMF Tone Service (Private Data Version 4 and Later)	4-130
■ Selective Listening Hold Service (Private Data Version 5 and Later)	4-137
■ Selective Listening Retrieve Service (Private Data Version 5 and Later)	4-143
■ Single Step Conference Call Service (Private Data Version 5 and Later)	4-148
■ Transfer Call Service	4-157

5	Set Feature Service Group	5-1
■	Overview	5-1
■	Set Advice of Charge Service (Private Data Version 5 and Later)	5-2
■	Set Agent State Service	5-6
■	Set Billing Rate Service (Private Data Version 5 and Later)	5-18
■	Set Do Not Disturb Feature Service	5-23

Contents

- Set Forwarding Feature Service 5-26
- Set Message Waiting Indicator (MWI) Feature Service 5-31

6	Query Service Group	6-1
	■ Overview	6-1
	■ Query ACD Split Service	6-2
	■ Query Agent Login Service	6-6
	■ Query Agent State Service	6-13
	■ Query Call Classifier Service	6-22
	■ Query Device Info	6-25
	■ Query Device Name Service	6-32
	■ Query Do Not Disturb Service	6-41
	■ Query Forwarding Service	6-43
	■ Query Message Waiting Service	6-46
	■ Query Station Status Service	6-50
	■ Query Time Of Day Service	6-53
	■ Query Trunk Group Service	6-56
	■ Query Universal Call ID Service (Private)	6-60

7	Snapshot Service Group	7-1
	■ Overview	7-1
	■ Snapshot Call Service	7-2
	■ Snapshot Device Service	7-6

8	Monitor Service Group	8-1
	■ Overview	8-1
	Change Monitor Filter Service	
	— cstaChangeMonitorFilter()	8-1
	Monitor Call Service — cstaMonitorCall()	8-1

Contents

Monitor Calls Via Device Service	
— cstaMonitorCallsViaDevice	8-2
Monitor Device Service — cstaMonitorDevice()	8-2
Monitor Ended Event — CSTAMonitorEndedEvent	8-2
Monitor Stop On Call Service (Private)	
— attMonitorStopOnCall()	8-2
Monitor Stop Service — cstaMonitorStop()	8-3
Event Filters and Monitor Services	8-3
Local Connection Info and Monitor Services	8-5
■ Change Monitor Filter Service	8-6
■ Monitor Call Service	8-13
■ Monitor Calls Via Device Service	8-22
■ Monitor Device Service	8-31
■ Monitor Ended Event Report	8-40
■ Monitor Stop On Call Service (Private)	8-42
■ Monitor Stop Service	8-46

9	Event Report Service Group	9-1
■	CSTAEventCause and LocalConnectionState	9-1
	Event Minimization Feature on G3 PBX	9-2
■	Call Cleared Event	9-3
■	Charge Advice Event (Private)	9-8
■	Conferenced Event	9-13
■	Connection Cleared Event	9-32
■	Delivered Event	9-39
■	Diverted Event	9-73
■	Entered Digits Event (Private)	9-77
■	Established Event	9-80
■	Failed Event	9-109
■	Held Event	9-114
■	Logged Off Event	9-116
■	Logged On Event	9-119
■	Network Reached Event	9-122

Contents

■ Originated Event	9-129
■ Queued Event	9-135
■ Retrieved Event	9-139
■ Service Initiated Event	9-142
■ Transferred Event	9-146
■ Event Report Detailed Information	9-165
Analog Sets	9-165
Redirection	9-165
Redirection on No Answer	9-165
Switch Hook Operation	9-165
ANI Screen Pop Application Requirements	9-166
Announcements	9-167
Answer Supervision	9-167
Attendants and Attendant Groups	9-167
Attendant Specific Button Operation	9-168
Attendant Auto-Manual Splitting	9-168
Attendant Call Waiting	9-168
Attendant Control of Trunk Group Access	9-169
AUDIX	9-169
Automatic Call Distribution (ACD)	9-169
Announcements	9-169
Interflow	9-169
Night Service	9-170
Service Observing	9-170
Auto-Available Split	9-170
Bridged Call Appearance	9-170
Busy Verification of Terminals	9-171
Call Coverage	9-171
Call Coverage Path Containing VDNs	9-172
Call Forwarding All Calls	9-172
Call Park	9-172
Call Pickup	9-173
Call Vectoring	9-173
Call Prompting	9-175
Lookahead Interflow	9-175
Multiple Split Queueing	9-175
Call Waiting	9-175

Contents

Conference	9-176
Consult	9-176
CTI Link Failure	9-176
Data Calls	9-176
DCS	9-177
Direct Agent Calling and Number of Calls In Queue	9-177
Drop Button Operation	9-177
Expert Agent Selection (EAS)	9-177
Logical Agents	9-177
Hold	9-178
Integrated Services Digital Network (ISDN)	9-178
Multiple Split Queueing	9-178
Personal Central Office Line (PCOL)	9-179
Primary Rate Interface (PRI)	9-179
Ringback Queueing	9-180
Send All Calls (SAC)	9-180
Service-Observing	9-180
Temporary Bridged Appearances	9-180
Terminating Extension Group (TEG)	9-181
Transfer	9-181
Trunk-to-Trunk Transfer	9-181

10	Routing Service Group	10-1
	■ Overview	10-1
	■ Route End Event	10-2
	■ Route End Service (TSAPI Version 2)	10-6
	■ Route End Service (TSAPI Version 1)	10-10
	■ Route Register Abort Event	10-12
	■ Route Register Cancel Service	10-14
	■ Route Register Service	10-17
	■ Route Request Service (TSAPI Version 2)	10-20
	■ Route Request Service (TSAPI Version 1)	10-36
	■ Route Select Service (TSAPI Version 2)	10-39

Contents

■ Route Select Service (TSAPI Version 1)	10-49
■ Route Used Event (TSAPI Version 2)	10-51
■ Route Used Event (TSAPI Version 1)	10-54

11	System Status Service Group	11-1
■	Overview	11-1
	System Status Request Service — cstaSysStatReq()	11-1
	System Status Start Service — cstaSysStatStart()	11-1
	System Status Stop Service — cstaSysStatStop()	11-1
	Change System Status Filter Service — cstaChangeSysStatFilter()	11-2
	System Status Event — CSTASysStatEvent	11-2
	System Status Events — Not Supported	11-2
■	System Status Request Service	11-3
■	System Status Start Service	11-10
■	System Status Stop Service	11-19
■	Change System Status Filter Service	11-21
■	System Status Event	11-29

A	Enhanced Voice Terminal Display	37
----------	--	----

IN	Index	IN-1
-----------	--------------	------

Contents

Issue 1 — December 2001

Introduction

1

Purpose and Scope

This document provides application developers with detailed information about the Computer- Supported Telecommunications Applications (CSTA) services and the Telephony Services Application Programming Interface (TSAPI) for the DEFINITY® Enterprise Communications Server (ECS) Generic 3 (G3) PBX in a Telephony Services environment.

The following information is included:

- Supported CSTA Services and TSAPI functions
- Parameter semantics
- Interactions with G3 PBX features
- TSAPI syntax
- G3 private services, private events, and private data syntax (services and functions provided by a specific switch vendor in addition to CSTA definitions).

The Telephony Services (Tserver) and the G3 PBX Driver (G3PD) software were originally developed for the NetWare® environment. Later they were developed for the Windows NT® environment. Information in this document applies to Microsoft® Windows® and the Windows NT platform as well as for the NetWare environment. Support for the NetWare environment was only provided up to Release 2.2.

Intended Audience

This document is intended for Telephony Services application developers programming in a DEFINITY G3 PBX environment. The document assumes familiarity with CSTA architecture and services defined in Technical Report ECMA TR/52 and Standard ECMA-179. It also assumes familiarity with DEFINITY G3 PBX features and procedures as described in the documents contained in the DEFINITY ECS Release 8 Documentation Library, 555-230-833.

Terminology

NOTE:

The following terms are used in this document: “DEFINITY Generic 3” or “Generic 3” for DEFINITY Communications System Generic 3, and “G3PD” for the DEFINITY Generic 3 PBX Driver. The terms “PBX” and “switch” are used interchangeably to mean “private branch exchange.” The phrase “SECURITY ALERT” warns you of possible security and toll fraud issues.

The definitions below explain some important terms. More detailed definitions appear in context when key concepts, functions, and services are described.

API Control Services (ACS)

An application uses a subset of TSAPI known as ACS to open, close, and control a communication channel (stream) to a Telephony Server (Tserver). Once a stream is open, the application may use the stream to request CSTA services from the Tserver.

EXE, DLL, and NLM

An EXE is an executable program and a DLL is a Dynamic Link Library software module on the Microsoft Windows/Windows NT platform. An NLM is a NetWare Loadable Module in the NetWare environment. Table 1-1 shows the software (i.e., the Tserver and the G3PD), the platform, and the associated file extensions for each.

Table 1-1. Platforms and Associated File Extensions

Software	Platform	File Extension
Telephony Services (Tserver)	Windows NT	EXE
Telephony Services (Tserver)	NetWare	NLM
G3 PBX Driver (G3PD)	Windows NT	DLL
G3 PBX Driver (G3PD)	NetWare	NLM

G3 CSTA Services

These services consist of the features and functions available to applications using the DEFINITY Generic 3 (G3) PBX together with the G3 PBX Driver (G3PD) software. The functions include those CSTA Services supported by the G3 PBX as well as private G3 PBX services. The G3 CSTA 3.30 implementation supports DEFINITY Communications System PBXs of version 8 or later for G3i, G3r, and G3s (Premier Business Package [PBP] only). While the latest version G3 PBXs support all the private services, earlier G3 PBXs do not.

This document defines the set of G3 CSTA Services.

G3 PBX Driver (G3PD)

The G3 PBX Driver, abbreviated as G3PD, is a Dynamic Link Library (DLL) on a Microsoft Windows NT Server. The G3PD software communicates with both the DEFINITY G3 PBX and the Tserver to provide switch services to Telephony Services applications.

PBX Driver

A switch-dependent Dynamic Link Library (DLL) that provides vendor-specific telephony services to the client applications. This DLL is provided by the vendor supplying the PBX and CSTA services for that switch.

Private Data

A mechanism that allows a switch vendor to provide value-added services that go beyond those defined in CSTA. The G3PD provides a number of private data services (for example, switch-collected call prompter digits in events or detection of answering machines on predictive calls). Private data features are specific to a given switch manufacturer.

Private data is available in the `privateData` parameter, which is an option in all CSTA requests, responses, and events. This document defines a C structure for use with the G3 PBX that overlays the `privateData` parameter in those request, response, and event messages where the G3 PBX provides or accepts private data. In addition, the G3 PBX provides several services not defined by CSTA. In these cases, CSTA Escape Services carry private data to provide these services.

Telephony Services API (TSAPI)

The Telephony Services C language definition of CSTA services, data types (parameters and structures), and event messages that enable applications to access Telephony Services. TSAPI is switch-independent and supports any Telephony Services-compliant driver. TSAPI includes CSTA Escape Services and private data allowing PBX vendors to provide value-added services within TSAPI.

Telephony Services (Tserver)

The software module that provides communication between telephony-enabled applications and the PBX driver. For convenience, an instance of the Telephony Services EXE software running on a particular server is referred to as a Tserver. The Tserver is responsible for processing the open and close requests of communications channels for all clients connected to the local area network (LAN). The Telephony Services EXE (on a Microsoft Windows NT machine) routes messages from the PBX driver to applications waiting for telephony events and passes the messages received from applications (TSAPI Service Requests) to the PBX driver. The Telephony Services EXE enforces user restrictions administered in the Tserver's Security Database (SDB). The Tserver is switch-independent and supports all Telephony Services PBX drivers.

Conventions

Please familiarize yourself with the following terms since they appear often in this document.

Party, Connection, ConnectionID

These terms, which are used interchangeably, all refer to a telephony user * a human, an application, or another resource such as a port on a voice-response unit.

Client, Client Application, Application, Process

These terms, which are used interchangeably, refer to a TSAPI application, a cooperative process distributed between a switching function and a computing function.

Related Documents

For a list of related DEFINITY and Centre Vu Computer Telephony documents, see the Preface ("About This Document") of *CentreVu Computer Telephony, Telephony Services and CallVisor PC Installation* (INSTALL.PDF). Following is a list of documents specifically related to CSTA services.

CSTA Services Documents

Following is a list of documents related to CSTA services:

- *Standard ECMA-179 Services For Computer Supported Telecommunications Applications (CSTA), European Computer Manufacturers Association, June 1992*

Defines CSTA services for Open Systems Interconnection (OSI) Layer 7 communication between a computing network and a telecommunications network. This standard, plus its companion Standard ECMA-180, reflects agreements of ECMA member companies on the first phase of CSTA standards. Optional reading for a G3 CSTA application developer.

- *Standard ECMA-180 Protocol For Computer Supported Telecommunications Applications (CSTA), European Computer Manufacturers Association, June 1992*

Defines a CSTA protocol for OSI Layer 7 communication between a computing network and a telecommunications network. Optional reading for a G3 CSTA application developer.

- *Computer Supported Telecommunications Applications ECMA TR/52, European Computer Manufacturers Association, June 1990*

Foundation for the OSI Layer 7 service protocol message interface communication between computing applications and switching applications. Recommended reading for a G3 CSTA application system engineer who designs the CSTA application.

- *Telephony Services Application Programming Interface (TSAPI), Version 2*

The Telephony Services Application Programming Interface (TSAPI) provides a programming environment that may be used with any PBX for which there is a CentreVu Computer-Telephony PBX driver. This document defines the CSTA application programming interface specification for the CentreVu Computer-Telephony platform. It defines the TSAPI programming interface and provides a tutorial on the CSTA client-server operational model. It is required reading for a CTI application developer.

This document is included on the CVCT CD-ROM as TSAPI.PDF.

New Features for Private Data Version 6

Release 3.30 of the G3PD provides an expanded set of private data features called private data version 6. If your DEFINITY switch supports these new features, you will be able to use them in your API. Most of these new features were introduced in DEFINITY G3V7 or G3V8.

See Table 3-3 (Private Data Summary) in the “Private Data Version Control” section within Chapter 3 for a complete list of private data features and the associated switch versions that support them.

Customer Support

For questions about Telephony Services, Tserver operation, or the DEFINITY G3 PBX Driver, call **1-800-344-9670** and follow the voice prompts for Call Center Solutions (CentreVu Products) and then for PassageWay® Computer-Telephony Integration.

How to Comment on This Document

Avaya welcomes your comments on this document. Please complete the reader comment card at the end of this document and return it.

You may send additional comments to:

Avaya, Inc.
Avaya University
Call Center Segment
Room 2G-528
101 Crawfords Corner Rd.
Holmdel, NJ 07733-3030

Purpose of CSTA

The Computer Supported Telecommunications Applications (CSTA) standard lets computer applications control switch devices, monitor switch devices, route calls, activate switch features, and integrate with switch functions in a variety of ways. Switches and computers can connect using Computer-Telephony Integration (CTI) technology.

When a computer and a switch are connected by CTI, each can then use the other's services. The computer might run an application that controls call distribution. Conversely, the switch might run an application that uses a database management system available on the computer. Neither system could implement these capabilities independently. CSTA provides an architectural framework that enables switch and computer systems to enhance network capabilities using CTI technology.

CSTA Standard and API Specification

Standard ECMA-179 defines the CSTA services. *Standard ECMA-180* defines the application protocol data units for the CSTA services. Together, these documents form the basis of the Telephony Services CSTA implementation. They provide switch and computer vendors with a specification that supports the same set of telephony services. However, the CSTA standards documents do not provide programming specifics. *Telephony Services Application Programming Interface (TSAPI)* provides the necessary programming specifications for the Windows NT and NetWare Telephony Services products.

Each service described in ECMA-179 has a corresponding CSTA Service in TSAPI. Each TSAPI function specifies the application program syntax for the corresponding CSTA.

A vendor may provide more services than are defined in the CSTA standard. CSTA provides a “private data” mechanism that allows for the support of vendor-specific services. Conversely, there are various optional protocol elements defined in the CSTA standard that a vendor may not provide. One important purpose of this guide is to specify which optional elements the G3 PBX does and does not support. In addition, the guide defines the programming interface for the G3 private data and private services.

CSTA Communication Layers Model

A CSTA application resides partially on a computer (providing data management) and partially on a switch (providing Telephony Services). The CSTA protocol defines the communications interface between the application layers of the switch and computer. The CSTA standard defines an OSI application layer communication protocol that provides a client/server relationship between the computer and the switch. The low-level communication protocol (“Lower Layer Interconnection System”) is implementation-dependent and is transparent to the application layer services.

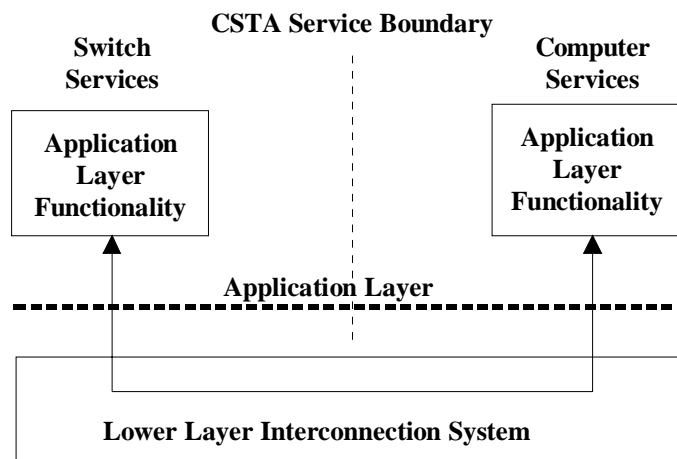


Figure 2-1. CSTA Communications Model

CSTA Client/Server Operational Model

The CSTA standard defines a client/server operational model as a relationship between two peer application layer processes where one process is able to perform a service for another. A “server” process performs a service for a “client” process.¹ In Telephony Services, a client application typically makes requests to manipulate various telecommunication objects on a switch.

The CSTA client/server relationship allows for bi-directional services. Both switching and computer applications can assume the role of either client or server. The computer or switch application that makes a request for service is a client. The switch or computer application that provides the service is the server. Currently, Routing Service is the only CSTA service in which the switch application is the client. In all other CSTA services, the computer application is the client.

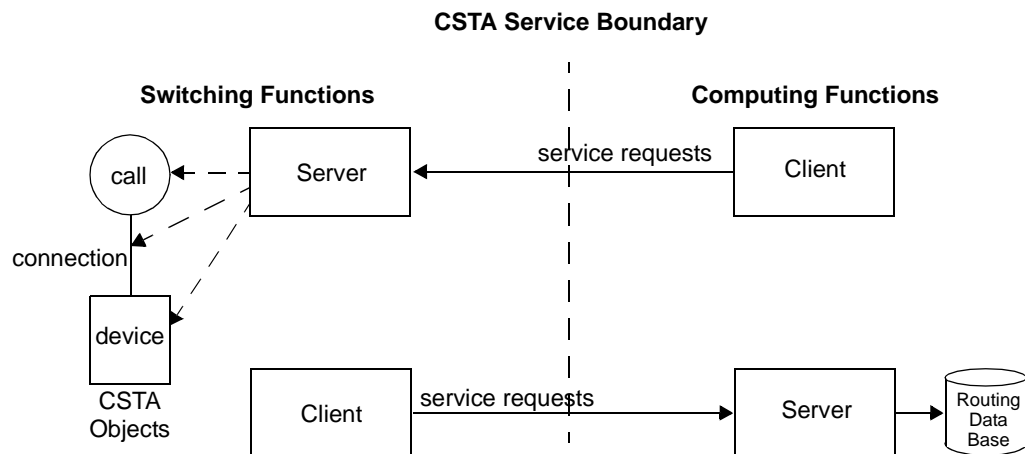


Figure 2-2. CSTA Architecture: Bi-directional Services

When an application requests a service, a local communications component in the client communicates the request to the server. Each instance of a request creates a new client/server relationship.

1. In this document, the terms “application” and “process” are used interchangeably.

CSTA Client/Server Session and Operation Invocation Model

A client must establish a communication channel to a Tserver before it can request service from that Tserver. In Telephony Services, this communication channel is an API Control Service stream. This stream establishes a session between a TSAPI application (at a client PC) and the server. An application uses the `acsOpenStream` function to open a stream. The function returns an `acsHandle` that the application uses to identify the stream. At that time, the application may use the `invokeID` in some other request.

When a client application requests a CSTA Service, it passes an `invokeID` that it may use later to associate a response from the server with a specific request. A client's request for service is also called an *operation invocation*. The server replies (via a *service response*) to the client's request with either confirmation (result) or failure (error/rejection) and includes the `invokeID` in the response.

Some services (such as monitoring a call or device) continue their operation beyond the service response. Since the `invokeID` no longer identifies the service invocation after an acknowledgment, an additional identifier is necessary for such services. These services return a `cross-referenceID` in their acknowledgment. The `cross-referenceID` is a unique value that an application can use to associate event reports with the initiating service request. The cross-reference terminates when the service stops.

Call Control and Call Events

CSTA Call Control Services allow a client application to control a call or connections on a switch.

Although client applications can manipulate switch objects, Call Control Services do not provide CSTA Event Reports as objects change state. To monitor switch object state changes (that is, to receive CSTA Event Report Services from a switch), a client must request a CSTA Monitor Service for an object before it requests Call Control Services for that object.

G3 CSTA System Overview

In a G3 CSTA environment, the G3 PBX is connected to a Tserver via one or more CTI links. The CTI links can terminate on a Basic Rate Interface (BRI) circuit pack (PC/ISDN Card) in a 386/486/Pentium® (or later) NetWare Telephony Server, or on an Ethernet adapter card on a 386/486/Pentium (or later) Windows NT or NetWare Telephony Server.

⇒ **NOTE:**

NOTE: The PC/ISDN Card does not work on a 100 MHz or higher speed computer and it is not supported on a Windows NT Tserver. Desktop PCs connect to the Windows NT or the NetWare Tserver on a LAN. Voice terminals connect directly to the G3 PBX and can be analog, digital, or ISDN.

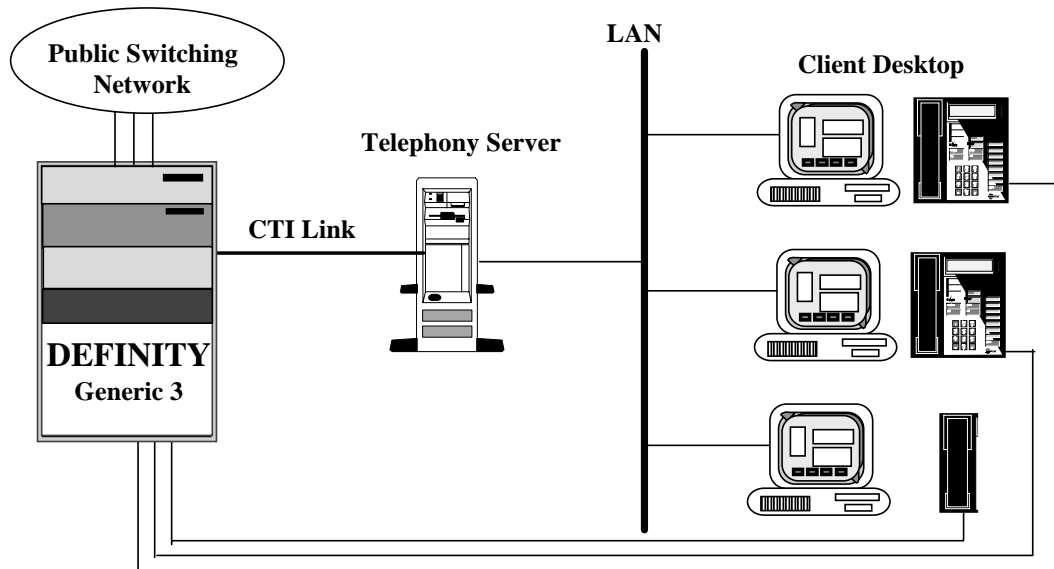


Figure 2-3. CSTA System

G3 CSTA Software Overview

The G3 PBX software and the G3 PBX Driver (G3PD) DLL/NLM running on a Windows NT or NetWare Tserver provide G3 CSTA services. The G3PD software passes the CSTA requests and responses between client applications and the call processing software on the G3 PBX. The CTI link between the G3 PBX and the Tserver provides the interface for the G3 CSTA functions.

The client application uses a TSAPI library. This library is available for various desktop PC operating system environments. The client application uses the TSAPI to establish a CSTA session with a Tserver EXE or NLM on the Tserver.

The Tserver EXE/NLM manages the communication between client applications and the G3PD.

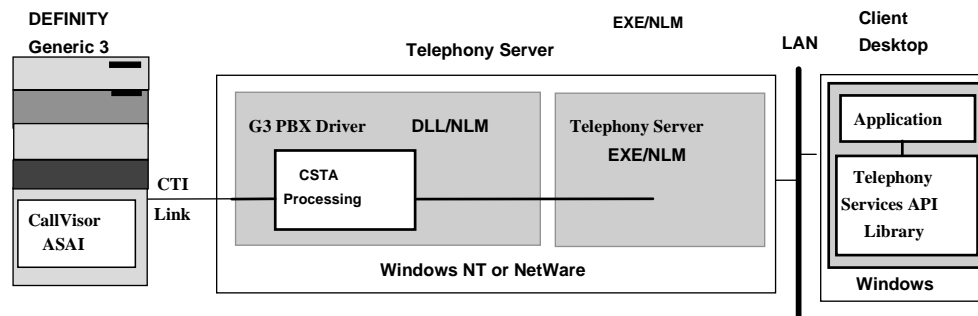


Figure 2-4. Communications Between Client PCs and the DEFINITY G3PD

DEFINITY Generic 3 PBX Driver (G3PD)

A PBX driver is a *switch-dependent* Windows NT DLL or NetWare NLM that provides vendor- specific Telephony Services to client applications. The Windows NT or NetWare Telephony Services product provides a G3 PBX Driver DLL or NLM for the DEFINITY Generic 3 PBX switches.

A G3 PBX Driver communicates with both a DEFINITY PBX and the Tserver EXE/NLM. The G3PD communicates with the G3 PBX via an interface known as a CTI link with the CTI protocol stack software. The CTI link connectivity is supported by either a BRI link (NetWare platform only) or an Ethernet LAN. A major function of the G3PD is to translate between the CSTA protocol and the G3 PBX CTI link protocol.

⇒ NOTE:

The number of CTI links available for your system depends on the version and type of platform being used.

The main functions of the G3 PBX Driver DLL/NLM are as follows:

- to handle CSTA telephony requests from the Tserver EXE/NLM, translate them into the appropriate G3 protocol requests, and send the protocol requests to a G3 PBX
- to communicate with the G3 PBX
- to handle requests, responses, and events from the G3 PBX, translate them into corresponding CSTA messages, and send the messages to the Tserver EXE/NLM
- to provide administration and maintenance for the G3 CTI link

When the G3PD is loaded, it initializes the CTI protocol stack with the G3 PBX. The G3PD then registers its services with the Tserver. After establishing communications to the Tserver, the G3PD then begins to service application requests.

For most CSTA services, the G3 switch is the server. However, for the Routing Service, the client/server relationship reverses. In this case, the Telephony Services application becomes a routing server.

A single G3PD can support a maximum of eight CTI links. On the NetWare platform, the maximum number of BRI links is four and the total BRI and LAN links cannot be more than eight. Each BRI link requires a separate hardware interface card.

Release 2 G3 PBX Driver on NetWare or a Release 3 G3 PBX Driver on Windows NT provides multiple links and multiple PBX functionality using the following configurations:

- multiple ISDN BRI (NetWare only) or LAN CTI links, or both, between a single G3 switch and the G3PD
- multiple links between a single G3PD and a maximum of eight different G3 switches, each with its own links

G3 Private Data Library Overview

For each G3 private service and each CSTA Service that contains private data, a private data function is defined and provided in the ATTPRIV.DLL for client applications on each client platform. If the private service is an extension of a defined CSTA Service, the private data function sets up the private parameters in the private data portion of the standard CSTA Service request. If the private service is not defined in the CSTA standard, the CSTA Escape Service is used for the service request and the private data function sets up the private service type and parameters in the private data portion of the Escape Service request. Any private parameters of a service request confirmation are provided in the private data portion of the confirmation event. A private event is sent to the client via the CSTA Private Status Event. The event type and associated parameters are provided in the private data portion of the CSTA Private Status Event.

The G3 PBX Private Data API interface (function calls, data structures, etc.) follows the TSAPI programming interface model. This document describes the G3 private data functions and their parameters.

TSAPI Software Components Overview

This section describes the TSAPI Library, the Telephony Server (Tserver) EXE/NLM, the DEFINITY Generic 3 PBX Driver (G3PD), and multiple CTI links to the G3PD.

TSAPI Library

The TSAPI Library provides the CSTA Services to applications running on client workstations, Windows NT, or NetWare servers. Application software can use the library to communicate with a G3 switch.

The client TSAPI Library comprises various communication layers. These layers are responsible for:

- providing the API Control Services and the CSTA API
- encoding/decoding data to and from a client operating system (OS) independent byte stream
- sending/receiving data from the network

Telephony Server (Tserver) EXE/NLM

The Tserver is a *switch-independent* Windows NT EXE or NetWare NLM that runs on a server connected to a LAN. The Tserver processes the open and close stream requests for all clients, thereby managing communications sessions (ACS streams) between client applications and PBX drivers.

The Tserver also authenticates client service requests, checks for permissions in the SDB, and passes requests to the appropriate PBX driver. The Tserver returns the PBX Driver responses to the requesting client.

PBX drivers register their services with the Tserver. In turn, the Tserver advertises these services to clients or places this information in an SDB. When a driver unregisters, the Tserver no longer advertises the driver's services to clients.

The Tserver uses the SDB to authenticate the user privileges for various devices and Telephony Services combinations such as controlling a given phone, controlling a call, monitoring a given device, querying, and routing incoming calls to a given device. When an application opens a stream on behalf of a user, the Tserver verifies that the user has a valid login.

Administration and maintenance capabilities are also available from the Tserver.

While only one Tserver can be loaded onto a Windows NT or NetWare server, the Tserver is capable of supporting multiple PBX drivers.

Multiple CTI Links to a G3PD

Release 3 G3PD supports up to eight CTI links for Windows NT. Multiple LAN CTI links can run on the same physical Ethernet link.

Release 2 G3PD supports up to eight CTI links for Novell® NetWare. Among these links, a maximum of four can be PC/ISDN links and a maximum of eight can be LAN CTI links. PC/ISDN links and LAN CTI links can be used at the same time, but the maximum of PC/ISDN links and LAN CTI links is eight. Each PC/ISDN link requires a physical BRI link and PC/ISDN Card hardware, but multiple LAN CTI links can run on the same physical Ethernet link.

These CTI links can go to as many as eight different G3 PBXs, can all go to the same PBX, or can be grouped in any other manner.

Having multiple CTI links to a G3 PBX provides the following advantages over a single link:

- throughput — multiple links can service more requests per unit of time than a single link
- redundancy — if one link goes out of service, others still provide service to the G3 PBX
- load balancing — G3PD balances the load among multiple links. The balancing is as even as the mapping from TSAPI to the G3 CTI link protocols will permit.

The following diagrams show some of the possible ways that four CTI links can connect from a G3PD to one or more G3 PBXs.

⇒ NOTE:

Although a G3PD supports multiple CTI links, there is a limit of one G3PD per Tserver.

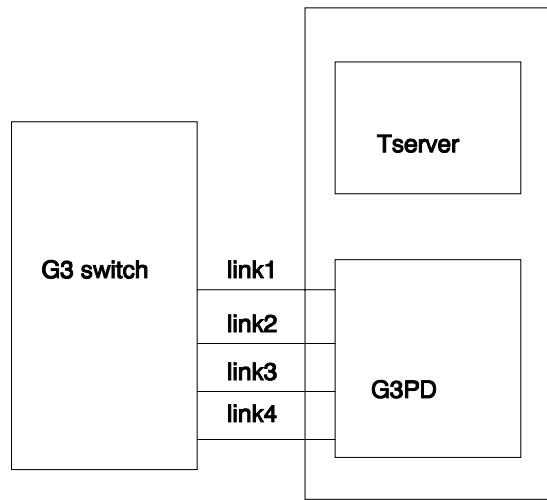


Figure 2-5. Example 1: Multiple CTI Link Connections — One Switch

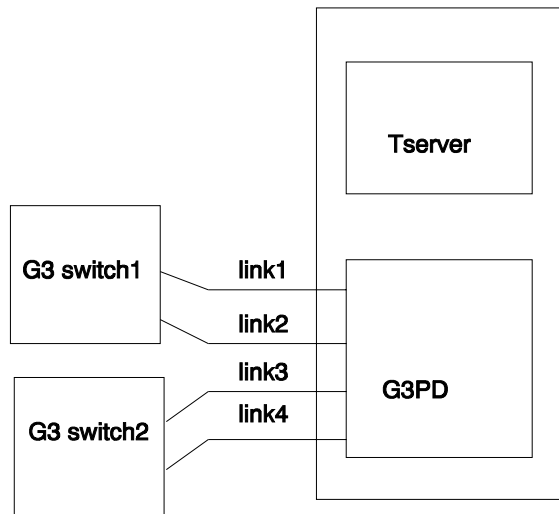


Figure 2-6. Example 2: Multiple CTI Link Connections — Two Switches

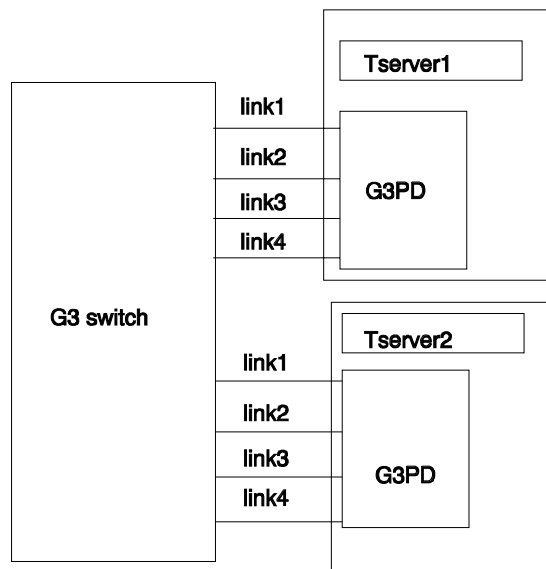


Figure 2-7. Example 3: Multiple CTI Link Connections — Two G3PDs and Tservers

When a G3PD has multiple CTI links to one or more G3 PBXs, each set of links to a different PBX must register a separate service with the Tserver EXE/NLM. If there are multiple links to any one PBX, then the G3PD will register at least one service, and optionally more than one; but no more than one service per CTI link. The G3PD behavior in this case is specified when the G3PD is installed. Both options are useful. Aggregating the links in a single registered service effectively provides wider bandwidth for a single CTI link.

Progress and Status of the CTI Link on NetWare

The G3PD NLM (not on Windows NT) reports the progress and status of each CTI link on the system console. The messages are as follows:

- **G3PD: Initializing Board(s). Please Wait!**

This message indicates that the G3PD is pumping the PC/ISDN communication board(s) that provide(s) CTI connectivity to the G3 switch. The G3PD will not display this message for a system with all CTI links over an Ethernet LAN.

- **G3PD: Connecting CTI Link(s). Please Wait!**

G3PD is attempting to establish the CTI connectivity to the G3 switch.

- **G3PD: CTI Link 1 is up and establishing service with the switch. Please Wait!**

Communication Layer 3 is established to the switch and waiting for the Application Layer to come up.

- **G3PD: CTI Link 1 is connected to a G3V4 switch.**

The indicated CTI link is connected to a G3V4 (or other version) switch. All CTI links connected to the same switch should report the same switch version in this message.

- **G3PD: CTI Link 1 is in service.**

The Application Layer is up and the G3PD can send messages to and receive messages from the switch via the indicated CTI link. This is the last message for a working CTI link.

- **G3PD WARNING: CTI LINK 2 IS NOT UP.**

Communication Layer 3 of the indicated CTI link is not established to the switch. Check the link hardware, switch administration, and G3PD.INI for link parameters. This will be the last message for a failed CTI link. If the problem persists, the G3PD will keep trying to bring the link up. When the problem is resolved and the link comes up and is in service, progress and status messages will be displayed.

⇒ NOTE:

Any change to G3PD.INI requires reloading the G3PD to bring the G3PD.INI into effect.

- **G3PD WARNING: CTI LINK 3 IS NOT IN SERVICE.
G3PD WARNING: Please check switch administration or link connection.**

Communication Layer 3 of the indicated CTI link is established to the switch, but the Application Layer is not up. The G3PD cannot send messages to or receive messages from the switch via the indicated CTI link. Check switch administration of the link or link connection. This will be the last message for a failed CTI link. If the problem persists, the link needs to be taken down (taken off-line via G3OAM, busied out via switch administration, or unplugged) and then brought up (brought on-line via G3OAM, unbusied by switch administration, or plugged in) again, before it can be put in service. When the link comes up and is in service, progress and status messages will be displayed.

- **G3PD WARNING: CTI Link 1 is DOWN.**

The indicated CTI link is down (the link was up before) and out of service. The G3PD will try to bring the link up again. When it comes up and is in service, progress and status messages will be displayed.



NOTE:

For NetWare, the G3PD.DLL does not provide a system console display; therefore, these messages cannot be viewed when using the G3PD.DLL alone.

Progress and Status of the CTI Link on Windows NT

The G3PD.DLL reports the progress and status of each CTI link in the error log (errlog.txt or g3pdlog.txt, depending on the registry setting). The following is an example of messages in error log recording progress and status (1 link only) between G3PD.DLL start (at 16:47:40) and end (stop at 17:57:54).

```
3/30/98 16:47:40 G3PD 17 2619 WARNING: ===== G3PD: Started.   Elvis is ... =====
3/30/98 16:47:40 G3PD 58256 0 AUDIT_TRAIL: g3pdInitProtocolStack has started.
3/30/98 16:47:40 G3PD 58256 0 AUDIT_TRAIL: asai mtce server thread id 265
3/30/98 16:47:40 G3PD 58256 0 AUDIT_TRAIL: oam server thread id 147
3/30/98 16:47:40 G3PD 6 2203 AUDIT_TRAIL: [main]: PBX_PING'S_SWITCH creation succeeded.
3/30/98 16:47:40 G3PD 17234 220 AUDIT_TRAIL: [PBX_PING'S_SWITCH]: TDI Driver Register succeeded -
524288 bytes (HWM=419430 bytes)
3/30/98 16:47:41 G3PD 61012 0 AUDIT_TRAIL: Poll_maint_server: Sent Heartbeat Confirmation to switch
3/30/98 16:47:42 G3PD 290 2017 WARNING: [SLO.CPP]: CTI Link 1 is UP.
3/30/98 16:47:43 G3PD 921 2020 WARNING: [SLO.CPP]: CTI Link 1 is a Version 3 link.
3/30/98 16:47:43 G3PD 1665 2015 WARNING: [SLO.CPP]: CTI Link 1 is in service.
3/30/98 16:47:43 G3PD 1666 2019 WARNING: [SLO.CPP]: CTI Link 1 is connected to a G3V6 switch.
3/30/98 17:57:54 G3PD 17233 215 AUDIT_TRAIL: [PBX-AUD_PING'S_SWITCH]: PBX thread terminating
3/30/98 17:57:55 G3PD 17233 215 AUDIT_TRAIL: [PBX-SAN_PING'S_SWITCH]: PBX thread terminating
3/30/98 17:57:56 G3PD 58261 0 AUDIT_TRAIL: g3pdUnloadProtocolStack called
3/30/98 17:57:58 G3PD 61010 0 WARNING: Closing the hb server.
3/30/98 17:57:58 G3PD 61010 0 WARNING: Closing the hb server.
3/30/98 16:48:00 G3PD 58262 0 AUDIT_TRAIL: g3pdUnloadProtocolStack complete
3/30/98 16:48:00 G3PD 25002 401 AUDIT_TRAIL: [G3Handler]: Received Signal SIGTERM - Unloading
G3PD
3/30/98 16:48:00 G3PD 18 2620 WARNING: ===== G3PD: Unloaded.   ... dead! =====
```

Figure 2-8. Sample Messages in Error Log

G3 CSTA Service Groups

3

Supported Services and Service Groups

DEFINITY G3 CSTA Services Release 3.30 supports the service groups defined in Table 3-1 on DEFINITY Generic 3 PBXs. Services that are not supported are listed in Table 3-2.

Table 3-1. Supported CSTA Services

Service Group	Service Group Definition	Supported Service(s)
Call Control	The services in this group enable a telephony client application to control a call or connection on the G3 PBX. Typical uses of these services are placing calls from a device and controlling a connection for a single call.	<ul style="list-style-type: none"> ■ Alternate Call ■ Answer Call ■ Clear Call ■ Clear Connection ■ Conference Call ■ Consultation Call ■ Consultation-Direct-Agent Call (private) ■ Consultation Supervisor-Assist Call (private) ■ Deflect Call ■ Hold Call ■ Make Call ■ Make Direct-Agent Call (private) ■ Make Predictive Call ■ Make Supervisor-Assist Call (private) ■ Pickup Call ■ Reconnect Call ■ Retrieve Call ■ Selective Listening Hold (private V5) ■ Selective Listening Retrieve (private V5) ■ Send DTMF Tone (private) ■ Single Step Conference (private V5) ■ Transfer Call

Table 3-1. Supported CSTA Services

Service Group	Service Group Definition	Supported Service(s)
Set Feature	The services in this group allow a client application to set switch-controlled features or values on a G3 PBX device.	<ul style="list-style-type: none"> ■ Set Advice Of Charge (private V5) ■ Set Agent State ■ Set Bill Rate (private V5) ■ Set Do Not Disturb ■ Set Forwarding ■ Set Message Waiting Indicator
Query	The services in this group allow a client to query device features and static attributes of a G3 PBX device.	<ul style="list-style-type: none"> ■ Query ACD Split (private) ■ Query Agent Login (private) ■ Query Agent Measurements (private) ■ Query Agent State ■ Query Call Classifier (private) ■ Query Device Info ■ Query Device Name ■ Query Do Not Disturb ■ Query Forwarding ■ Query Message Waiting Indicator ■ Query Split/Skill Measurements (private) ■ Query Time of Day (private) ■ Query Trunk Group (private) ■ Query Trunk Group Measurements (private) ■ Query Station Status (private) ■ Query Universal Call ID (private V5) ■ Query VDN Measurements (private)
Snapshot	The services in this group allow a client application to take a snapshot of a call or device on a G3 PBX.	<ul style="list-style-type: none"> ■ Snapshot Call ■ Snapshot Device

Table 3-1. Supported CSTA Services

Service Group	Service Group Definition	Supported Service(s)
Monitor	The services in this group allow a client application to request and cancel the reporting of events that cause a change in the state of a G3 PBX object.	<ul style="list-style-type: none"> ■ Change Monitor Filter ■ Monitor Call ■ Monitor Calls Via Device ■ Monitor Device ■ Monitor Ended Event ■ Monitor Stop on Call (private) ■ Monitor Stop
Event Report	The services in this group provide a client application with the reports of events that cause a change in the state of a call, a connection, or a device.	<p>Call Event Reports:</p> <ul style="list-style-type: none"> ■ Call Cleared ■ Charge Advice (private V5) ■ Connection Cleared ■ Conferenced ■ Delivered ■ Diverted ■ Entered Digits (private) ■ Established ■ Failed ■ Held ■ Network Reached ■ Originated ■ Queued ■ Retrieved ■ Service Initiated ■ Transferred <p>Agent State Event Reports:</p> <ul style="list-style-type: none"> ■ Logged On ■ Logged Off

Table 3-1. Supported CSTA Services

Service Group	Service Group Definition	Supported Service(s)
Routing	The services in this group allow a G3 PBX to request and receive routing instructions for a call from a client application.	<ul style="list-style-type: none"> ■ Route End Event ■ Route End Service ■ Route Register Abort Event ■ Route Register Cancel Service ■ Route Register Service ■ Route Request Service ■ Route Select Service ■ Route Used Event
Escape	The services in this group allow an application to request a private service that is not defined by the CSTA Standard.	<ul style="list-style-type: none"> ■ Escape Service ■ Private Event ■ Private Status Event
Maintenance	The services in this group allow an application to request (1) device status maintenance events that provide status information for device objects, and (2) bi-directional system status maintenance services that provide information on the overall status of the system.	None
System Status	The services in this group allow an application to request system status information from the G3PD.	<ul style="list-style-type: none"> ■ System Status Request ■ System Status Start ■ System Status Stop ■ Change System Status Filter ■ System Status Event

Table 3-2. Unsupported CSTA Services

Service Group	Unsupported Service(s) or Event Report(s)
Call Control	<ul style="list-style-type: none"> ■ Group Pickup Call
Set Feature	None
Query	<ul style="list-style-type: none"> ■ Query Last Number
Snapshot	None

Table 3-2. Unsupported CSTA Services

Service Group	Unsupported Service(s) or Event Report(s)
Monitor	None
Event Reports	Call Event Reports: None Agent State Event Reports: <ul style="list-style-type: none"> ■ Not Ready Event ■ Ready Event ■ Work Not Ready Event ■ Work Ready Event Feature Event Reports: <ul style="list-style-type: none"> ■ Call Info Event ■ Do Not Disturb Event ■ Forwarding Event ■ Message Waiting Event
Routing	<ul style="list-style-type: none"> ■ Re-Route Event
Escape	<ul style="list-style-type: none"> ■ Send Private Event
Maintenance	<ul style="list-style-type: none"> ■ Back in Service Event ■ Out of Service Event
System Status	<ul style="list-style-type: none"> ■ System Status Request Event ■ System Status Ended Event ■ System Status Event Send

Transferring or Conferencing a Call Together with Screen Pop Information

Many desktop applications involve scenarios where an incoming call arrives at a monitored phone, (e.g., a claims agent) and the application uses caller information to pop a screen at that desktop. At some point, the claims agent realizes that both the call and the data screen need to be shared with some other person, (e.g., a supervisor). The claims agent may need to conference in the supervisor, or may need to transfer the call to the supervisor. In both cases, a similar application running at the supervisor's desktop that is monitoring the supervisor's phone needs to obtain information about the original caller from CSTA events to pop the same screen at the supervisor's desktop.

Before designing a screen pop application, an application designer must first understand the caller information that G3PD makes available. When an incoming call arrives at a monitored station device, the G3PD provides CSTA Delivered and Established events that contain a variety of caller information:

- Calling Number (CSTA parameter) - This parameter contains the calling number, when known. An application may use the calling number to access customer records in a database. The Event Report chapter contains detailed information about the facilities that provide Calling Number.
- Called Number (CSTA parameter) - This parameter contains the called number, when known. Often this parameter contains the "DNIS" for an incoming call from the public network. An application may use the called number to pop an appropriate screen when, for example, callers dial different numbers to order different products.
- Digits Collected by Call Prompting (G3 private data) - Integrated systems often route callers to a voice response unit that collects the caller's account number. These voice response units can often be integrated with a G3 PBX so that the caller's account number is made available to the monitoring application. An application may use the collected digits to access customer records in a database.
- User-to-User Information (UUI) (G3 private data) - This parameter contains information that some other application has associated with the incoming call. UUI has the important property that it can be passed across certain facilities (PRI) which can be purchased within the public switched network. An application may use the calling number to access customer records in a database.
- Lookahead Interflow Information (G3 private data) - This parameter contains information about the call history of an incoming call that is being forwarded from a remote G3 PBX.

CSTA Services Used to Conference or Transfer Calls

Application designers next need to understand the various CSTA services that might be used to conference or transfer calls and the different event contents that result from these services.

There are two sequences of TSAPI services that an application may use to conference or transfer calls:

- the sequence
 1. CSTAConsultationCall;
 2. CSTAConferenceCall or CSTATransferCall.

The CSTAConsultationCall service places an active call on hold and then makes a consultation call (such as the call to the supervisor in the example above). The CSTAConferenceCall or CSTATransferCall conferences or transfers the call.

The unique (and important!) attribute of CSTAConsultationCall is that the consultation service associates the call being placed on hold with the consultation call.

An application monitoring the phone receiving the consultation call will see information about the original caller in a G3 private data item called "Original Call Information" appearing in the CSTA Delivered event.

"Original Call Information" gives an application (such as the supervisor's) the information necessary to pop a screen using the original caller's information at the time that the call begins alerting at the consultation desktop.

⇒ NOTE:

Applications that need to pass information about the original caller and have a screen pop when the call alerts at the consultation desktop should use the CSTAConsultationCall service to place those calls.

- the sequence
 1. CSTAHold;
 2. CSTAMakeCall;
 3. CSTAConferenceCall or CSTATransferCall.

This sequence of operations emulates what a user might do manually at a phone.

Unlike the CSTAConsultationCall service, these operations do not associate any information about the call being placed on hold with the call that is being made. In fact, such an association cannot be made because the calling station may have multiple calls on hold and the G3PD cannot anticipate which of those will actually be transferred.

However, using this sequence of operations does, in some cases, pass information about the original caller in events for the consultation call. This occurs for transferred calls when the transferring party hangs up before the consultation call is answered. This is known as a "blind transfer".

Notice that when the consultation party answers the blind transfer, there are two parties on the call, the original caller and the consultation party. Therefore, when the calling party answers, G3PD puts information about the original caller in the CSTA Established event. This sequence allows an application monitoring the party receiving the consultation call to pop a screen about the original caller only in the case of a blind transfer and only when the call is answered.

Using Original Call Information to Pop a Screen

When an incoming call arrives at a monitored desktop (the claims agent in the example), an application can use any of the caller information listed earlier to pop a screen. When the application uses CSTAConsultationCall to pass a call to another phone, the G3PD retains the original caller information in a block of private data called "Original Call Information". The G3PD passes "Original Call Information" in the Delivered and Established events for the consultation call. Thus, an application monitoring the consultation desktop can use any of the original caller information to pop a screen.

Application designers must be aware that:

- CSTAConsultationCall is the recommended way of passing calls from desktop to desktop in such a way that the original caller information is available for popping screens.
- G3PD shares "Original Call Information" with applications using that G3PD to monitor phones.
- "Original Call Information" cannot be shared across different G3PDs or shared with other G3 PBX CTI platforms.

When applications use "Original Call Information" to pop screens, the applications monitoring phones for the community of users among which calls are transferred (typically call center or service center agents) must use the same G3PD.

- An application designer using "Original Call Information" must make sure that the system is configured so that applications running on behalf of those users that may pass calls to one another all monitor the user phones through the same G3PD.

- G3PD shifts information into the "Original Call Information" block as the call information changes. For example, since prompted digits don't change because a call is transferred, the original prompted digits may be in the prompted digit private data parameter rather than the "Original Call Information" block.
- Applications using caller information should look first in the "Original Call Information" block, and if they find nothing there, use the information in the other private data and CSTA parameters.

Note that, for example, if a call passes through monitored VDN A (which collects digits) and then passes through monitored VDN B (which also collects digits) and then is delivered to monitored VDN C, then in the Delivered event we find the digits from VDN A in the Original Call Information for the call and the digits from VDN B in the Collected Digits private data for the call.



NOTE:

Using this approach, the application will always use the original caller's information to pop the screen regardless of whether it is running at the desktop that first receives the call (the claims agent) or a consultation desktop (the supervisor's desktop).

Using UUI to Pass Information to Remote Applications

In addition to providing "Original Call Information" to allow original caller information to pass among applications using the same G3PD, the G3 PBX provides advanced private data features that let an application developer implement an application that passes caller information to applications:

- monitoring stations using different Tservers
- monitoring stations using another type of G3 CTI platform
- residing on a CTI platform at a remote G3 switch that is monitoring stations on that remote G3.

Since the G3 PBX associates User-to-User Information (UUI) with a call within the PBX, the G3 PBX makes the UUI for a call available on all of its CTI links. Further, when a G3 PBX supplies UUI when making a call (such as a consultation call) across PRI facilities in the public switched network, the UUI passes across the public network to the remote G3. The remote G3 then makes this UUI available to applications on its CTI links.

While "Original Call Information" is a way of sharing all caller information across applications using a given G3PD, UUI is the way to share information across a broader CTI application community, including applications running at remote switch sites.

An important decision in the design of an application that works across multiple G3PDs, CTI platforms, and remote G3 switches is what information passes between applications in the UUI.

Application designers must be aware of the following:

- Unlike "Original Call Information", the amount of information that UUI carries is limited.
Often the UUI is an account number that has been collected by a voice response unit or obtained from a customer database. It might also be the caller's telephone number. It might be a record or transaction identifier that the application defines.
- In all cases, the application is responsible for copying or entering the information into the call's UUI. Applications may enter information into a call's UUI when they make a call, route a call, or drop a call.
- When an application enters information into a call's UUI, any previous UUI is overwritten.
- Applications running on other G3PD, CTI platforms, or remote switches must all be designed to expect the same information in the UUI (and in the same format!). Obviously, application design for a system that spans multiple G3PDs, CTI platforms, or G3 switches must be coordinated.
- When an application encompasses users both within a G3PD and on other G3PDs, CTI platforms, or switches, then the application designer might use a hybrid approach that combines the best of "Original Call Information" (all of the original caller data) with the advantages of UUI (sharing information across G3 CTI links and remote switches).

TSAPI Version Control

As TSAPI evolves over time, the changes are reflected in different "versions" of TSAPI. The TSAPI specification describes these changes. TSAPI has evolved between the availability of the Release 1 G3PD and Release 2 G3PD. An application uses version control to specify which TSAPI version(s) it can use.

When writing a new application, application developers should always use the latest TSAPI version. An application coded using TSAPI Version 1 will operate unchanged with a G3PD that provides TSAPI Version 2, so long as the application requests Version 1. New applications programmed from this manual should use TSAPI Version 2. An application written using the Release 1 G3PD Programmer's Guide that uses a Release 2 G3PD will automatically receive TSAPI Version 1 services.

An application supplies a list of TSAPI versions it can support in the `apiVer` parameter of the `acsOpenStream()` function. TSAPI provides the syntax specification for `apiVer`. Applications requesting service from a Release 2 G3PD that can use either TSAPI version 1 or 2 might encode `apiVer` as "TS1-2". An

application requiring TSAPI version 2 should encode apiVer as "TS2". Applications that require TSAPI version 1 should encode apiVer as "TS1"¹. When an application makes a request that allows multiple versions, the application needs to check the value of the apiVer field in the response to determine which version to use on the stream.

See Table 3-1 in this chapter for a list of the services available to an application that opens a TSAPI stream. Where version 1 and version 2 of such a service are available (certain routing services), such an application must use the version that it requested.

Private Data Version Control

Just as TSAPI evolves over time, so may the private data services offered by a PBX driver. The G3PD driver typically supports multiple versions of private data. This is so existing applications do not have to be modified when the G3PD private data services change.

If an application wishes to utilize a new service offered in a particular private data version, then all private data services must be upgraded to the specified private data version. For example, if an application had previously been designed to work with private data version 4, it will work with a new release of the driver, but it will negotiate a private data version 4 stream. If a private data version 6 stream is negotiated, then all services must be upgraded to the private data version 6 format. A version 4 message that has been changed in version 6 is no longer supported on a version 6 stream. If the version 4 and version 6 format are the same, then that version 4 message will still be supported.

The private data version is independent of the TSAPI version. When an application opens a stream to a PBX driver, it may indicate to the PBX driver (i.e., the G3PD) which private data versions it supports. The confirmation event for the open stream request informs the application of the specific private data version that will be provided.

When opening a TSAPI version 2 stream, an application should provide a list of supported private data versions in the data portion of the private data buffer. The function attMakeVersionString() is provided to simplify formatting the G3PD version list.

The following code fragment illustrates how to format the private data buffer to request DEFINITY G3 (ATT) private data version 3, 4, 5 or 6:

-
1. An application written using the Release 1 TSAPI manual will encode apiVer to the #define CSTA_API_VERSION in csta.h. This approach results in TSAPI version 1 when the application is run with a release 2 G3PD. An application that is to be run with Release 1 and Release 2 G3PDs should encode apiVer using the "TS" format described in this manual.

```
ATTPrivateData_tprivateData;

/* ... */

/* Prepare the private data buffer for version request */
(void) strcpy(privateData.vendor, "VERSION");
privateData.data[0] = PRIVATE_DATA_ENCODING;

/* Request one of G3 private data versions 3 through 6 */
attMakeVersionString("3-6", &(privateData.data[1]));
privateData.length = strlen(&privateData.data[1]) + 2;
```

Applications designers should be aware of the following:

- The G3PD will select the highest private data version that it supports.
- The G3PD supports private data versions 2, 3, 4, 5 and 6 on a TSAPI version 2 stream.
- The open stream confirmation event will indicate both the TSAPI and the private data versions for the stream.
- The value of the private data version in the response may be lower than the version that the application requested. Therefore, the application should check that the private data version in the response is the value that it expects.
- Applications that open a TSAPI version 2 stream and that do not use any of the private data can request that the G3PD not supply any private data. This reduces traffic on the LAN. TSAPI describes how to open a version 2 TSAPI stream that will not supply any private data.
- Private data version control is not supported by TSAPI version 1. Applications that open TSAPI version 1 streams to the G3PD will always receive version 1 private data.

Private Data Version Feature Support

All G3 PBXs provide call prompting digits, the only private data item in version 1. Private data versions 2 through 6 encompass a much broader feature set, where some features may be dependent upon the switch version.

- Private data version 2 includes support for some features that are available only on the G3V3 and later releases.
- Private data versions 3 and 4 include support for some features that are available only with the G3V4 and later releases.
- Private data version 5 includes support for some features that are available only on the G3V5, G3V6, G3V7 and later releases.
- Private data version 6 includes support for some features that are available only on the G3V8 and later releases.

Before designing an application that uses a private data feature, see Table 3-3 (Private Data Summary) to ensure that the G3 PBX supports it.

CSTAGetAPICaps Confirmation Interface Structures for Private Data Versions 4, 5, and 6

Beginning with private data version 4, the G3PD provides the G3 PBX version-dependent private services in the CSTAGetAPICaps Confirmation private data interface, as defined by the following structures:

Private Data Version 5 and 6 Syntax

```
typedef struct ATTGetAPICapsConfEvent_t
{
char  switchVersion[16]; // specifies the switch
                                // version - G3V2, G3V3,
                                // G3V3, G3V4, G3V5,
                                // G3V6 or G3V8. (no new
                                // capabilities are pro-
                                // vided with G3V7 so the
                                // G3 PBX driver does not
                                // differentiate between
                                // a G3V6 and a G3V7.

Boolean  sendDTMFTone; // TRUE - supported,
                        // FALSE - not supported
Boolean  enteredDigitsEvent; // TRUE - supported,
                                // FALSE - not supported
Boolean  queryDeviceName; // TRUE - supported,
                                // FALSE - not supported
Boolean  queryAgentMeas; // TRUE - supported,
                                // FALSE - not supported
Boolean  querySplitSkillMeas; // TRUE - supported,
                                // FALSE - not supported
Boolean  queryTrunkGroupMeas; // TRUE - supported,
                                // FALSE - not supported
Boolean  queryVdnMeas; // TRUE - supported,
                                // FALSE - not supported
Boolean  singleStepConference; // TRUE - supported,
                                // FALSE - not supported
Boolean  selectiveListeningHold; // TRUE - supported,
                                // FALSE - not supported
Boolean  selectiveListeningRetrieve; // TRUE - supported
                                // FALSE - not supported
Boolean  setBillingRate; // TRUE - supported,
                                // FALSE - not supported
Boolean  queryUcid; // TRUE - supported,
                                // FALSE - not supported
Boolean  chargeAdviceEvent; // TRUE - supported,
                                // FALSE - not supported
}
```

```
Boolean reserved1;           // reserved for future use
Boolean reserved2;           // reserved for future use
} ATGetAPICapsConfEvent_t;
```

Private Data Version 4 Syntax

```
typedef struct ATTV4GetAPICapsConfEvent_t
{
    char    switchVersion[16]; // specifies the switch
                                // version - G3V2, G3V3,
                                // G3V3, G3V4, G3V5, or
                                // G3V6
    Boolean sendDTMFTone;      // TRUE - supported,
                                // FALSE - not supported
    Boolean enteredDigitsEvent; // TRUE - supported,
                                // FALSE - not supported
    Boolean queryDeviceName;   // TRUE - supported,
                                // FALSE - not supported
    Boolean queryAgentMeas;    // TRUE - supported,
                                // FALSE - not supported
    Boolean querySplitSkillMeas; // TRUE - supported,
                                // FALSE - not supported
    Boolean queryTrunkGroupMeas; // TRUE - supported,
                                // FALSE - not supported
    Boolean queryVdnMeas;      // TRUE - supported,
                                // FALSE - not supported
    Boolean reserved1;         // reserved for future use
    Boolean reserved2;         // reserved for future use
} ATTV4GetAPICapsConfEvent_t;
```

⇒ NOTE:

G3 PBX capabilities are obtained only once when the G3PD is loaded during negotiation with the switch. If G3PD is not unloaded and reloaded after the switch software version is changed (for example, from G3V3 to G3V4 or vice versa), then once this change is made, and the G3PD is not unloaded and reloaded again, the cstaGetAPICaps requests will return the capabilities that the G3PD obtained when it was first loaded. Thus cstaGetAPICaps will not reflect the real capabilities of the new switch version.

Private Data Feature Summary

Table 3-3 provides a complete list of private data features. The associated initial DEFINITY switch and G3PD releases that support each one are included, as well as the version of private data in which the feature was first introduced.

Table 3-3. Private Data Summary

Private Data Feature	Initial DEFINITY Release	Initial DEFINITY PBX Driver Release	Initial Private Data Version
Prompted Digits in Delivered Events	All	R2.1 (private data)	V1
Priority, Direct Agent, Supervisor Assist Calling	All	R2.1 (private data)	V2
Enhanced Call Classification	All	R2.1 (private data)	V2
Trunk, Classifier Queries	All	R2.1 (private data)	V2
LAI in Events	All	R2.1 (private data)	V2
Launching Predictive Calls from Split	All	R2.1 (private data)	V2
Application Integration with Expert Agent Selection	G3V3	R2.1 (private data)	V2
User-to-User Info (Reporting and Sending)	G3V3	R2.1 (private data)	V2
Multiple Notification Monitors (two on ACD/VDN)	G3V3	All	
Launching Predictive Calls from VDN	G3V3	R2.1	
Multiple Outstanding Route Requests for One Call	G3V3	R2.1	
Answering Machine Detection	G3V3	R2.1 (private data)	V2
Established Event for Non-ISDN Trunks	G3V3	All	
Provided Prompter Digits on Route Select	G3V3	R2.1 (private data)	V2
Requested Digit Selection	G3V3	R2.1 (private data)	V2
VDN Return Destination (Serial Calling)	G3V3	R2.1 (private data)	V2
Deflect Call	G3V4	R2.2	
Pickup Call	G3V4	R2.2	
Originated Event Report	G3V4	R2.2	V3
Agent Logon Event Report	G3V4	R2.2 (private data)	V3

Table 3-3. Private Data Summary

Private Data Feature	Initial DEFINITY Release	Initial DEFINITY PBX Driver Release	Initial Private Data Version
Reason for Redirection in Alerting Event Report	G3V4	R2.2 (private data)	V3
Agent, Split, Trunk, VDN Measurements Query	G3V4	R2.2 (private data)	V3
Device Name Query	G3V4	R2.2 (private data)	V3
Send DTMF Tone	G3V4	R2.2 (private data)	V3
Distributing Device in Conferenced, Delivered, Established, and Transferred Events	All	R2.2 (private data)	V4
G3 Private Capabilities in cstaGetAPICaps Confirmation Private Data	G3V3	R2.2 (private data)	V4
Support Detailed DeviceIDType_t in Events	G3V3	R3.10 (private data)	V5
Set Bill Rate	G3V4	R3.10 (private data)	V5
Flexible Billing in Delivered Event, Established Event, and Route Request	G3V4	R3.10 (private data)	V5
Call Originator Type in Delivered Event, Established Event, and Route Request	G3V4	R3.10 (private data)	V5
Selective Listening Hold	G3V5	R3.10 (private data)	V5
Selective Listening Retrieve	G3V5	R3.10 (private data)	V5
Set Advice of Charge	G3V5	R3.10 (private data)	V5
Charge Advice Event	G3V5	R3.10 (private data)	V5
Reason Code in Set Agent State, Query Agent State, and Logout Event	G3V5	R3.10 (private data)	V5
27-Character Display Query Device Name Confirmation	G3V5	R3.10 (private data)	V5
Unicode Device ID in Events	G3V6	R3.10 (private data)	V5
Trunk Group and Trunk Member Information in Network Reached Event	G3V6	R3.10 (private data)	V5

Issue 1 — December 2001

Table 3-3. Private Data Summary

Private Data Feature	Initial DEFINITY Release	Initial DEFINITY PBX Driver Release	Initial Private Data Version
Universal Call ID (UCID) in Events	G3V6	R3.10 (private data)	V5
Single Step Conference	G3V6	R3.10 (private data)	V5
Pending Work Mode and Pending Reason Code in Set Agent State and Query Agent State	G3V8	R3.30 (private data)	V6
Trunk Group and Trunk Member Information in Delivered Event and Established Event regardless of whether Calling Party is Available	G3V8	R3.30 (private data)	V6
Trunk Group Information in Route Request Events regardless of whether Calling Party is Available	G3V8	R3.30 (private data)	V6
Trunk Group Information for Every Party in Transferred Events and Conferenced Events	G3V8	R3.30 (private data)	V6
User-to-User Info (UUI) is increased from 32 to 96 bytes	G3V8	R3.30 (private data)	V6

Migration from Private Data Version 5 to Private Data Version 6

An existing application, without any changes, can still work with the new G3PD.DLL that supports private data V6 interface, but the application cannot open a private data V6 interface and request any private data V6 features.

To migrate an existing private data V5 application into the private data V6 environment (i.e., V6 SDK) the changes shown in Table 3-4 are required.

- The list of Protocol Data Units (PDUs) or structure members in column one represents the original V5 code that is affected by the V6 interface.
- In order for the V5 code to continue to operate as private data using the V6 interface, you must change the PDUs or structure members listed in column one in your code to the associated name listed in column two (i.e., The "ATT" portion of the name is changed to "ATTV5" for the PDUs while v5 is prepended in the case of structure members).
- The PDU code names or structure members listed in column three are identical to the original V5 code names; however, their definitions are changed in the header files for the V6 interface.

⇒ NOTE:

The private data library has a convention whereby PDU names for the most recent private data version are always "unqualified," that is, the names do not contain any indication of a particular private data version. When a new version of an existing PDU is introduced, the new PDU assumes the name of the old PDU, and the name of the old PDU is changed to reflect the last private data version for which it was valid. The same naming convention is used when introducing a new version of an existing data type or structure member.

Table 3-4. Migration of PDUs and Structure Members to Private Data Version 6

Original V5 PDU or Structure Member Name	Required Changes to V5 PDU or Structure Member Name for V6 Interface	New V6 PDU or Structure MemberName
ATT_QUERY_AGENT_STATE_CONF ATTQueryAgentStateConfEvent_t queryAgentState	ATTV5_QUERY_AGENT_STATE_CONF ATTV5QueryAgentStateConfEvent_t v5queryAgentState	ATT_QUERY_AGENT_STATE_CONF ATTQueryAgentStateConfEvent_t queryAgentState
ATT_SET_AGENT_STATE ATTSetAgentState_t setAgentStateReq	ATTV5_SET_AGENT_STATE ATTV5SetAgentState_t v5setAgentStateReq	ATT_SET_AGENT_STATE ATTSetAgentState_t setAgentStateReq
N/A	New for V6	ATT_SET_AGENT_STATE_CONF ATTSetAgentStateConfEvent_t
ATT_ROUTE_REQUEST ATTRouteRequestEvent_t	ATTV5_ROUTE_REQUEST ATTV5RouteRequestEvent_t	ATT_ROUTE_REQUEST ATTRouteRequestEvent_t
ATT_TRANSFERRED ATTTransferredEvent_t	ATTV5_TRANSFERRED ATTV5TransferredEvent_t	ATT_TRANSFERRED ATTTransferredEvent_t
ATT_CONFERENCED ATTConferencedEvent_t	ATTV5_CONFERENCED ATTV5ConferencedEvent_t	ATT_CONFERENCED ATTConferencedEvent_t
ATT_CLEAR_CONNECTION ATTClearConnection_t	ATTV5_CLEAR_CONNECTION ATTV5ClearConnection_t	ATT_CLEAR_CONNECTION ATTClearConnection_t
ATT_CONSULTATION_CALL ATTConsultationCall_t	ATTV5_CONSULTATION_CALL ATTV5ConsultationCall_t	ATT_CONSULTATION_CALL ATTConsultationCall_t
ATT_MAKE_CALL ATTMakeCall_t	ATTV5_MAKE_CALL ATTV5MakeCall_t	ATT_MAKE_CALL ATTMakeCall_t
ATT_DIRECT_AGENT_CALL ATTDirectAgentCall_t	ATTV5_DIRECT_AGENT_CALL ATTV5DirectAgentCall_t	ATT_DIRECT_AGENT_CALL ATTDirectAgentCall_t
ATT_MAKE_PREDICTIVE_CALL ATTMakePredictiveCall_t	ATTV5_MAKE_PREDICTIVE_CALL ATTV5MakePredictiveCall_t	ATT_MAKE_PREDICTIVE_CALL ATTMakePredictiveCall_t
ATT_SUPERVISOR_ASSIST_CALL ATTSupervisorAssistCall_t	ATTV5_SUPERVISOR_ASSIST_CALL ATTV5SupervisorAssistCall_t	ATT_SUPERVISOR_ASSIST_CALL ATTSupervisorAssistCall_t
ATT_RECONNECT_CALL ATTReconnectCall_t	ATTV5_RECONNECT_CALL ATTV5ReconnectCall_t	ATT_RECONNECT_CALL ATTReconnectCall_t
ATT_CONNECTION_CLEARED ATTConnectionClearedEvent_t	ATTV5_CONNECTION_CLEARED ATTV5ConnectionClearedEvent_t	ATT_CONNECTION_CLEARED ATTConnectionClearedEvent_t
ATT_ROUTE_SELECT ATTRouteSelect_t	ATTV5_ROUTE_SELECT ATTV5RouteSelect_t	ATT_ROUTE_SELECT ATTRouteSelect_t
ATT_DELIVERED ATTDeliveredEvent_t	ATTV5_DELIVERED ATTV5DeliveredEvent_t	ATT_DELIVERED ATTDeliveredEvent_t
ATT_ESTABLISHED ATTEstablishedEvent_t	ATTV5_ESTABLISHED ATTV5EstablishedEvent_t	ATT_ESTABLISHED ATTEstablishedEvent_t
ATT_ORIGINATED ATTOriginatedEvent_t	ATTV5_ORIGINATED ATTV5OriginatedEvent_t	ATT_ORIGINATED ATTOriginatedEvent_t

Private Data Function Changes

Please note that the following private data functions are changed between V5 and V6.

Set Agent State

```
// attSetAgentState() - Private Data V5 Interface

RetCode_t    attSetAgentStateExt(// old function name used in V5 API
    ATTPrivateData_t*attPrivateData,
    ATTWorkMode_t    workMode,
    long              reasonCode); // new parameter in V5 API

// attSetAgentStateExt() - Private Data V6 Interface

RetCode_t    attV6SetAgentState(// new function name used in V6 API
    ATTPrivateData_t*attPrivateData,
    ATTWorkMode_t    workMode,
    long              reasonCode,
    Boolean           enablePending);// new parameter in V6 API
```

Private Data Sample Code

Sample Code 1

```
#include <stdio.h>

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

/*
 * Make Direct Agent Call - from "12345" to ACD Agent extension "11111"
 * - ACD agent must be logged into split "22222"
 * - no User to User info
 * - not a priority call
 */

ACSHandle_t acsHandle;           // An opened ACS Stream Handle
InvokeID_t  invokeID = 1;        // Application generated
                                           // Invoke ID
DeviceID_t  calling = "12345";   // Call originator, an on-PBX
                                           // extension
DeviceID_t  called = "11111";   // Call destination, an ACD
                                           // Agent extension
DeviceID_t  split = "22222";    // ACD Agent is logged into
                                           // this split
Boolean     priorityCall = FALSE; // Not a priority call
RetCode_t   retcode;            // Return code for service
                                           // requests
CSTAEvent_t cstaEvent;         // CSTA event buffer
unsigned    short    eventBufSize; // CSTA event buffer size
ATTPrivateData_t privateData;   // ATT service request private
                                           // data buffer

    retcode = attDirectAgentCall(&privateData, &split, priorityCall,
                                NULL);

    if ( retcode < 0 ) {
        /* Some kind of failure, need to handle this ... */
    }

    retcode = cstaMakeCall(acsHandle, invokeID, &calling, &called,
(PrivateData_t *)&privateData);

    if (retcode != ACSPOSITIVE_ACK) {
        /* Some kind of failure, need to handle this ... */
    }
}
```

Sample Code 1 (Continued)

```
/* Make Call request succeeded. Wait for confirmation event. */

eventBufSize = sizeof(CSTAEvent_t);
privateData.length = ATT_MAX_PRIVATE_DATA;

retcode = acsGetEventBlock(acsHandle, (void *)&cstaEvent,
                          &eventBufSize, (PrivateData_t *)&privateData, NULL);

if (retcode != ACSPOSITIVE_ACK) {
/* Some kind of failure, need to handle this ... */
}

if ((cstaEvent.eventHeader.eventClass == CSTACONFIRMATION) &&
    (cstaEvent.eventHeader.eventType == CSTA_MAKE_CALL_CONF))
{
    if (cstaEvent.event.cstaConfirmation.invokeID == 1) {
/*
* Invoke ID matches, Make Call is confirmed.
*/
    }
}
}
```

Sample Code 2

```
#include <stdio.h>

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

/*
 * Set Agent State - Request to log in ACD Agent with initial work mode
 * "Auto-In".
 */

ACSHandle_t acsHandle;           // An opened ACS Stream Handle
InvokeID_t  invokeID = 1;        // Application generated
                                           // Invoke ID
DeviceID_t  device = "12345";    // Device associated with
                                           // ACD Agent
AgentMode_t agentMode = AM_LOG_IN; // Requested Agent Mode
AgentID_t   agentID = "01";      // Agent login identifier
AgentGroup_t agentGroup = "11111"; // ACD split to log Agent into
AgentPassword_t *agentPassword = NULL; // No password, i.e., not EAS
RetCode_t   retcode;             // Return Code for service
                                           // requests
CSTAEvent_t cstaEvent;          // CSTA event buffer
unsigned short eventBufSize;     // CSTA event buffer size
ATTPrivateData_t privateData;    // ATT service request private
// data buffer

retcode = attV6SetAgentState(&privateData, WM_AUTO_IN, 0, TRUE);

if (retcode < 0 ) {
    /* Some kind of failure, need to handle this ... */
}

retcode = cstaSetAgentState(acsHandle, invokeID, &device, agentMode,
    &agentID, &agentGroup, agentPassword,
    (PrivateData_t *)&privateData);

if (retcode != ACSPOSITIVE_ACK) {
    /* Some kind of failure, need to handle this ... */
}
}
```


Sample Code 2 (Continued)

```
/* Set Agent State request succeeded. Wait for confirmation event.*/

eventBufSize = sizeof(CSTAEvent_t);
privateData.length = ATT_MAX_PRIVATE_DATA;

retcode = acsGetEventBlock(acsHandle, (void *)&cstaEvent,
                          &eventBufSize, (PrivateData_t *)&privateData, NULL);

if (retcode != ACSPOSITIVE_ACK) {
    /* Some kind of failure, need to handle this ... */
}

if ((cstaEvent.eventHeader.eventClass == CSTACONFIRMATION) &&
    (cstaEvent.eventHeader.eventType == CSTA_SET_AGENT_STATE_CONF)) {
    if (cstaEvent.event.cstaConfirmation.invokeID == 1) {
        /*
         * Invoke ID matches, Set Agent State is confirmed.
         * Private data is returned in confirmation event.
         */
        if (privateData.length > 0) {
            /* Confirmation contains private data */

            if (attPrivateData(&privateData, &attEvent) !=
                ACSPOSITIVE_ACK) {
                /* Decoding error */
            }
            else { // See whether the requested change is pending or not
                ATTSetAgentStateConfEvent_t *setAgentStateConf ;
                SetAgentStateConf = &privateData.u.setAgentState;
                if (SetAgentStateConf->isPending == TRUE)
                    // The request is pending
            }
        }
    }
}
```

Sample Code 3

```
#include <stdio.h>

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

/*
 * Query ACD Split via cstaEscapeService()
 */

ACSHandle_t acsHandle;           // An opened ACS Stream Handle
InvokeID_t  invokeID = 1;        // Application generated
                                           // Invoke ID
DeviceID_t  deviceID = "12345"; // Device associated with
                                           // ACD split
RetCode_t   retcode;            // Return code for service
                                           // requests
CSTAEvent_t cstaEvent;          // CSTA event buffer
unsigned shorteventBufSize;     // CSTA event buffer size
ATTPrivateData_t privatedata;   // ATT private data buffer
ATTEvent_t  attEvent;           // ATT event buffer
ATTQueryAcdSplitConfEvent_t     // Query ACD Split confirmation
    *queryAcdSplitConf;        // event pointer

retcode = attQueryAcdSplit(&privatedata, &deviceID);

if (retcode < 0 ) {
    /* Some kind of failure, need to handle this ... */
}

retcode = cstaEscapeService(acsHandle, invokeID,
    (PrivateData_t *)&privatedata);

if (retcode != ACSPOSITIVE_ACK) {
    /* Some kind of failure, need to handle this ... */
}
```

Sample Code 3 (Continued)

```
/*
 * Now wait for confirmation event.
 *
 * To retrieve private data return parameters for Query ACD Split,
 * the application must specify a pointer to a private data buffer as
 * a parameter to either the acsGetEventBlock() or acsGetEventPoll()
 * request. Upon return, the application passes the address
 * to attPrivateData() for decoding.
 */

eventBufSize = sizeof(CSTAEvent_t);
privateData.length = ATT_MAX_PRIVATE_DATA;

retcode = acsGetEventBlock(acsHandle, (void *)&cstaEvent,
                          &eventBufSize, (PrivateData_t *)&privateData, NULL);

if (retcode != ACSPOSITIVE_ACK) {
    /* Some kind of failure, need to handle this ... */
}

if ((cstaEvent.eventHeader.eventClass == CSTACONFIRMATION) &&
    (cstaEvent.eventHeader.eventType == CSTA_ESCAPE_SVC_CONF)) {
    if (cstaEvent.event.cstaConfirmation.invokeID != 1) {
        /* Error - wrong invoke ID */
    }
    else if (privateData.length > 0) {
        /* Confirmation contains private data */
    }

    if (attPrivateData(&privateData, &attEvent) != ACSPOSITIVE_ACK) {
        /* Decoding error */
    }
    else if (attEvent.eventType == ATT_QUERY_ACD_SPLIT_CONF) {
        queryAcdSplitConf = (ATTQueryAcdSplitConfEvent_t *)
            &attEvent.u.queryAcdSplit;
    }
    }
    else {
        /* Error - no private data in confirmation event */
    }
}
```

G3 CSTA Objects

Figure 3-1 illustrates the three types of CSTA objects: Device, Call, and Connection.



Figure 3-1. CSTA Objects: Device, Call and Connection

CSTA Object: Device

The term Device refers to both physical devices (stations, trunks, etc.) and logical devices (VDNs or ACD splits) that are controlled via the switch. Each device is characterized by a set of attributes. These attributes define the manner in which an application may observe and manipulate a device. The set of device attributes consists of: Device Type, Device Class, and Device Identifier.

Device Type

Table 3-5 defines the most commonly used G3 CSTA devices and their types:

Table 3-5. CSTA Device Type Definitions

CSTA Type	Definition	G3 Object
Station	A traditional telephone device or an AWOH station extension (for phantom calls). ¹ A station is a physical unit of one or more buttons and one or more lines.	Station or extension on the G3 PBX
ACD Group	A mechanism that distributes calls within a switch.	VDN, ACD split, or hunt group in the G3 PBX

Table 3-5. CSTA Device Type Definitions

CSTA Type	Definition	G3 Object
Trunk	A device used to access other switches.	G3 trunk
Trunk Group	A group of trunks accessed using a single identifier.	G3 trunk group
Other	A type of device not defined by CSTA.	Announcement, CTI (ASAI), modem pool, etc.

1. For DEFINITY software release 6.3 and later, a call can be originated from an AWOH station or some group extensions (i.e., a plain [non-ACD] hunt group). This is termed a phantom call. Most calls that can be requested for a physical extension can also be requested for an AWOH station and the associated event will also be received. If the call is made on behalf of a group extension, this may not apply. For a detailed description of the phantom call switch feature, refer to CallVisor ASAI Technical Reference (555-230-220).

CSTA defines device types that G3PD Services do not use: ACD Group, button, button group, line, line group, operator, operator group, station group.

Device Class

Different classes of devices can be observed and manipulated within the G3 CSTA programming environment. Common G3 CSTA Device Classes include: voice and other. G3PD does not support service requests for the CSTA data and image classes. G3PD may return the data class in response to a query.

Device Identifier

Each device that can be observed and manipulated needs to be referenced across the CSTA Service boundary. Devices are identified using one or both of the following types of identifiers:

Static Device Identifier

A static device identifier is stable over time and remains both constant and unique between calls. The static device identifier is known by both the TSAPI application and the switch. G3 internal extensions are static device identifiers. These include extensions that uniquely identify any G3 devices such as stations or AWOH station extensions (for phantom calls), ACD splits, VDNs, and logical agent login IDs. Valid phone numbers for endpoints external to the G3 PBX are also static

device identifiers². The presence of a static device ID in an event does not necessarily mean that the device is directly connected to the switch.

⇒ NOTE:

If the called device specified in a CSTA Make Call Service request is not an internal endpoint, the device identifier reported in the event reports for that device on that call may not be the same. The called device specified in the CSTA Make Call Service is a dialing digit sequence and it may not represent a true device identifier. For example, trunk access code can be specified as part of the dialing digits in the called device parameter of a CSTA Make Call Service request. However, the trunk access code will not be part of the device identifier of the called device in the event reports of that call. In a DCS (Distributed Communications System) or SDN (Software Defined Network) environment, even if a true device identifier (i.e., no trunk access code in the called device parameter) of an external endpoint is specified for the called device in a CSTA Make Call Service request, the G3 switch may not use the same device identifier in the event reports for the called device.

Dynamic Device Identifier

When a call is connected through a trunk with an unknown device identifier, a dynamic trunk identifier is created for the purpose of identifying the external endpoint. This identifier is not like a static device identifier that an application can store in a database for later use. An off-PBX endpoint without a known static identifier³ has a trunk identifier. Note that a trunk identifier does not identify the actual trunk or trunk group to which the endpoint is connected. The actual trunk and trunk group information, if available, is provided in the private data.

In order to manipulate and monitor calls that cross a G3 PBX trunk interface, an application needs to use the trunk identifier. G3PD preserves trunk identifiers across conference and transfer operations. G3PD may use different dynamic identifiers to represent endpoints connected to the same actual trunk at different times. A trunk identifier is meaningful to an application only for the duration of a call and should not be retained and used at a later time like a phone number or a station extension. A call identifier and a trunk identifier can comprise a connection identifier. A trunk identifier has a prefix 'T' and a '#' within its identifier (for example, T538#1, T4893#2).

-
2. If applicable, access and authorization codes can be specified with the static device identifier for the called device parameter of the Make Call Service.
 3. An off-PBX endpoint of an ISDN call may have a known static identifier.

Device Identifier Syntax

```
typedef char          DeviceID_t[64];

typedef enum DeviceIDType_t {
    DEVICE_IDENTIFIER = 0,
    IMPLICIT_PUBLIC = 20,
    EXPLICIT_PUBLIC_UNKNOWN = 30,
    EXPLICIT_PUBLIC_INTERNATIONAL = 31,
    EXPLICIT_PUBLIC_NATIONAL = 32,
    EXPLICIT_PUBLIC_NETWORK_SPECIFIC = 33,
    EXPLICIT_PUBLIC_SUBSCRIBER = 34,
    EXPLICIT_PUBLIC_ABBREVIATED = 35,
    IMPLICIT_PRIVATE = 40,
    EXPLICIT_PRIVATE_UNKNOWN = 50,
    EXPLICIT_PRIVATE_LEVEL3_REGIONAL_NUMBER = 51,
    EXPLICIT_PRIVATE_LEVEL2_REGIONAL_NUMBER = 52,
    EXPLICIT_PRIVATE_LEVEL1_REGIONAL_NUMBER = 53,
    EXPLICIT_PRIVATE_PTN_SPECIFIC_NUMBER = 54,
    EXPLICIT_PRIVATE_LOCAL_NUMBER = 55,
    EXPLICIT_PRIVATE_ABBREVIATED = 56,
    OTHER_PLAN = 60,
    TRUNK_IDENTIFIER=70,
    TRUNK_GROUP_IDENTIFIER=71
} DeviceIDType_t;

typedef enum DeviceIDStatus_t {
    ID_PROVIDED = 0,
    ID_NOT_KNOWN = 1,
    ID_NOT_REQUIRED = 2
} DeviceIDStatus_t;

typedef struct ExtendedDeviceID_t {
    DeviceID_t      deviceID;
    DeviceIDType_t deviceIDType;
    DeviceIDStatus_tdeviceIDStatus;
} ExtendedDeviceID_t;

typedef ExtendedDeviceID_t CallingDeviceID_t;

typedef ExtendedDeviceID_t CalledDeviceID_t;

typedef ExtendedDeviceID_t SubjectDeviceID_t;

typedef ExtendedDeviceID_t RedirectionDeviceID_t;
```

CSTA Device ID Type (Private Data Version 4 and Earlier)

If an application opens an ACSOpenStream with private data version 4 and earlier, G3PD supports only a limited number of types of DeviceIDType_t for the deviceIDType parameter of an ExtendedDeviceID_t. The types supported are described in Table 3-6.

Table 3-6. CSTA Device Type and Status (Private Data Version 4 and Earlier)

DeviceIDType_t	ConnectionID_Device_t	DeviceIDStatus_t	Type of Devices
DEVICE_IDENTIFIER	STATIC_ID	ID_PROVIDED	Internal or external endpoints that have a known device identifier
TRUNK_IDENTIFIER	DYNAMIC_ID	ID_PROVIDED	Internal or external endpoints that do not have a known device identifier
EXPLICIT_PUBLIC_UNKNOWN		ID_NOT_KNOWN or ID_NOT_REQUIRED	

CSTA Device ID Type (with Private Data Version 5 and Later)

If an application opens an ACAOpenStream with private data version 5 and later, G3PD supports CSTA DeviceIDType_t based on information from the switch, network, or internal information. The DEVICE_IDENTIFIER and TRUNK_IDENTIFIER are no longer supported. More descriptive types are used. Normally if a call is connected through an ISDN interface or a private network supporting a private numbering plan, more accurate device ID type will be provided to the application.

- DEVICE_IDENTIFIER (0) — This type is no longer used for an ACSOpenStream with private data version 5 and later.
- IMPLICIT_PUBLIC (20) — There is no actual numbering and addressing information about this endpoint received from the network or switch. However, from the number of digits (7 or more digits) of the device identifier associated with this endpoint, it may be a public number. Prefix or escape digits may be present.
- EXPLICIT_PUBLIC_UNKNOWN (30) — There are two cases for this type:

- There is no actual numbering and addressing information about this endpoint received from the network or switch. The device identifier is also unknown for this endpoint. An external endpoint without a known device identifier is most likely to have this type.
- The numbering and addressing information are provided by the ISDN interface from the network and G3 PBX to which the call is connected, but the network and switch has no knowledge of the type of the number (e.g., international, national, or local) of the endpoint. Prefix or escape digits may be present.
- **EXPLICIT_PUBLIC_INTERNATIONAL (31)** — This endpoint has an international number. The numbering plan and addressing type information are provided by the ISDN interface from the network and G3 PBX to which the call is connected. Prefix or escape digits shall not be included.
- **EXPLICIT_PUBLIC_NATIONAL (32)** — This endpoint has a national number. The numbering plan and addressing type information are provided by the ISDN interface from the network and G3 PBX to which the call is connected. Prefix or escape digits shall not be included.
- **EXPLICIT_PUBLIC_NETWORK_SPECIFIC (33)** — This endpoint has a network specific number. The numbering plan and addressing type information are provided by the ISDN interface from the network and G3 PBX to which the call is connected. The type of network specific number is used to indicate the administration/service number specific to the serving network, (e.g., used to access an operator).
- **EXPLICIT_PUBLIC_SUBSCRIBER (34)** — This endpoint has a network specific number. The numbering plan and addressing type information are provided by the ISDN interface from the network and G3 PBX to which the call is connected. Prefix or escape digits shall not be included.
- **EXPLICIT_PUBLIC_ABBREVIATED (35)** — This endpoint has an abbreviated number. The numbering and addressing information are provided by the ISDN interface from the network and G3 PBX to which the call is connected.
- **IMPLICIT_PRIVATE (40)** — There is no actual numbering plan and addressing type information about this endpoint received from the network or switch. However, from the number of digits (6 or less digits) of the device identifier associated with this endpoint, it may be a private number. Prefix or escape digits may be present. An internal endpoint or an external endpoint across the DCS or private network may have this type. Note that it is not unusual for an internal endpoint's type changing from **IMPLICIT_PRIVATE** to **EXPLICIT_PRIVATE_LOCAL_NUMBER** when more information about the endpoint is received from the switch.
- **EXPLICIT_PRIVATE_UNKNOWN (50)** — This endpoint has a private numbering plan and the addressing type is unknown. An endpoint is unlikely to have this device ID type.

- EXPLICIT_PRIVATE_LEVEL3_REGIONAL_NUMBER (51) — This endpoint has a private numbering plan and its addressing type is level 3 regional. An endpoint is unlikely to have this device ID type.
- EXPLICIT_PRIVATE_LEVEL2_REGIONAL_NUMBER (52) — This endpoint has a private numbering plan and its addressing type is level 2 regional. An endpoint is unlikely to have this device ID type.
- EXPLICIT_PRIVATE_LEVEL1_REGIONAL_NUMBER (53) — This endpoint has a private numbering plan and its addressing type is level 1 regional. An endpoint is unlikely to have this device ID type.
- EXPLICIT_PRIVATE_PTN_SPECIFIC_NUMBER (54) — This endpoint has a private numbering plan and its addressing type is PTN specific. An endpoint is unlikely to have this device ID type.
- EXPLICIT_PRIVATE_LOCAL_NUMBER (55) — There are two cases for this type:
 - There is no actual numbering plan and addressing type information about this endpoint received from the switch or network. However, this endpoint has a device identifier and its type is identified by the G3PD call processing as a local number or a local endpoint to the G3 PBX. A local endpoint is one that is directly connected to the G3 PBX to which the G3PD is connected. An endpoint that is not directly connected to the G3 PBX to which the G3PD is connected, but can be accessed through the DCS or private network that connects to the G3 PBX to which the G3PD is connected is not a local endpoint. A G3PD local endpoint normally either has a type of EXPLICIT_PRIVATE_LOCAL_NUMBER or IMPLICIT_PRIVATE. Note that it is not unusual for an endpoint's type changing from IMPLICIT_PRIVATE to EXPLICIT_PRIVATE_LOCAL_NUMBER when more information above the endpoint is received from the switch. An internal endpoint is most likely to have this device ID type with this case.
 - This endpoint has a private numbering plan and its addressing type is local number. An endpoint is unlikely to have this device ID type with this case.
- EXPLICIT_PRIVATE_ABBREVIATED (56) — This endpoint has a private numbering plan and its addressing type is abbreviated. An endpoint is unlikely to have this device ID type.
- OTHER_PLAN (60) — This endpoint has a type "none of the above." An endpoint is unlikely to have this type.
- TRUNK_IDENTIFIER (70) — This type is no longer used for an ACSOpenStream with private data version 5 and later.
- TRUNK_GROUP_IDENTIFIER (71) — This type is not used by G3PD.

CSTA Object: Call

Applications can use TSAPI to control and monitor Call behavior, including establishment and release. There are two types of call attributes: Call Identifier and Call State.

Call Identifier (callID)

When a call is initiated, the G3 switch allocates a unique Call Identifier (callID). Before a call terminates, it may progress through many different states involving a variety of devices. Although the call identifier may change (as with transfer and conference, for example), its status as a CSTA object remains the same. A callID first becomes visible to an application when it appears in an event report or confirmation event. The allocation of a callID is always reported. Each callID is specified in a connection identifier parameter.

⇒ NOTE:

The TSAPI interface passes callID parameters within connectionID parameters.

Call Identifier Syntax

```
typedef struct ConnectionID_t {
    long          callID;           // always specified in a
                                   // connectionID
    DeviceID_t    deviceID;        // set to 0, when only callID
                                   // is interested
    ConnectionID_Device_t devIDType; // STATIC_ID or DYNAMIC_ID
} ConnectionID_t;
```

Call State

A "call state" is a descriptor (initiated, queued, etc.) that characterizes the state of a call. Even though a call may assume several different states throughout its duration, it can only be in a single state at any given time. The set of connection states comprises all of the possible states a call may assume. Call state is returned by the Snapshot Device Service for devices that have calls.

CSTA Object: Connection

A CSTA "connection" is a relationship that exists between a call and a device. Many API Services (Hold Call Service, Retrieve Call Service, and Clear Call Service, for example) observe and manipulate connections. Connections have the following attributes:

Connection Identifier (connectionID)

A connectionID is a combination of Call Identifier (callID) and Device Identifier (deviceID). The connectionID is unique within a G3 PBX. An application cannot use a connectionID until it has received it from the G3PD. This rule prevents an application from fabricating a connectionID.

A connectionID always contains a callID value. A G3PD connectionID may contain a static or dynamic (for Trunk ID) device identifier. If the callID is the only value that is present, the deviceID is set to 0 (with DYNAMIC_ID). The callID of a connectionID assigned to an endpoint on a call may change when the call is transferred or conferenced, but the deviceID of the connectionID assigned to an endpoint will not change when the call is transferred or conferenced.

For a call, there are as many Connection Identifiers as there are devices on the call. For a device, there are as many Connection Identifiers as there are calls at that device.

Connection Identifier Conflict

A device may connect to a call twice. This can happen for external endpoints with the same calling number from an ISDN network or from an internal device with different line appearances connected to the same call. In these rare cases, the G3PD resolves the device identifier conflict in the connection identifiers by replacing one of the device identifiers with a trunk identifier when two calls that have the same device (this is not the device conferencing the call) on them are merged by a call conference or transfer operation.

⇒ NOTE:

The connection identifier of a device on a call can change in this case.

Connection Identifier Syntax

```
typedef char          DeviceID_t[64];

typedef enum ConnectionID_Device_t {
    STATIC_ID = 0,
    DYNAMIC_ID = 1
} ConnectionID_Device_t;

typedef struct ConnectionID_t {
    long                callID;
    DeviceID_t          deviceID;
    ConnectionID_Device_t devIDType;
} ConnectionID_t;
```

Connection State

A connection state is a descriptor (initiated, queued, etc.) that characterizes the state of a single CSTA connection. Connection states are reported by Snapshots taken of calls or devices. Changes in connection states are reported as event reports by Monitor Services.

Figure 3-2 illustrates a connection state model that shows typical connection state changes. This connection state model derives from the CSTA connection state model. It provides an abstract view of various call state transitions that can occur when a call is either initiated from, or delivered to, a device. Note that it does not include all the possible states that may result from interactions with G3 PBX features. The G3 PBX also incorporates state transitions that may not be shown.

⇒ NOTE:

It is strongly recommended that applications be event driven. Being state driven, rather than event driven, may result in an unexpected state transition that the program has not anticipated. This often occurs because some party on the call invokes a G3 feature that interacts with the call in a way that is not part of a typical call flow. The diagram that follows captures only typical call state transitions. The G3 PBX has a large number of specialized features that interact with calls in many ways.

This model does not represent a complete programming model for the call state/event report/connection state relationship.

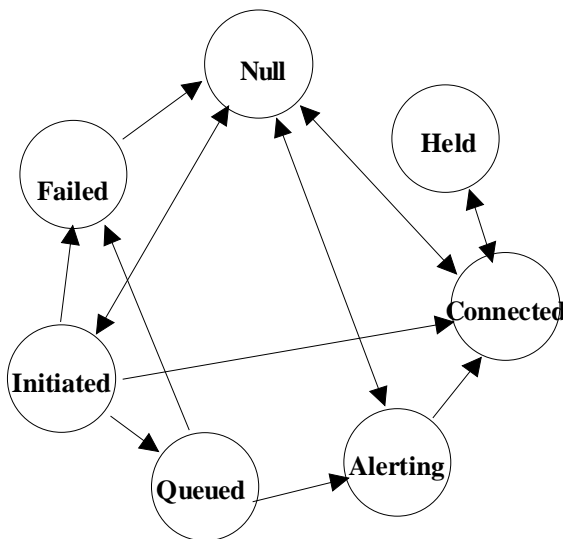


Figure 3-2. Sample Connection State Model

In Figure 3-2, circles represent connection states. Arrows are used to signify transitions between states. A transition from one connection state to another results in the generation of an event report. The various connection states are defined as follows:

Table 3-7. Connection State Definitions

Definition	Description
Null	No relationship exists between the call and device; a device does not participate in a call.
Initiated	A device is requesting service. Usually, this results in the creation of a call. Often, this is when a station receives a dial tone and begins to dial.
Alerting	A device is alerting (ringing). A call is attempting to become connected to a device. The term "active" is also used to indicate an alerting (or connected) state.
Connected	A device is actively participating in a call, either logically or physically (that is, not Held). The term "active" is also used to indicate a connected (or alerting) state.
Held	A device inactively participates in a call. That is, the device participates logically but not physically.
Queued	Normal state progression has been stalled. Generally, either a device is trying to establish a connection with a call or a call is trying to establish a connection with a device.
Failed	Normal state progression has been aborted. Generally, either a device is trying to establish a connection with a call or a call is trying to establish a connection with a device. A Failed state can result from a failure to connect to the calling device (origin) or to the called device (destination). A Failed state can also be caused by a failure to create the call or other factors.
Unknown	A device participates in a call, but its state is not known.

Table 3-7. Connection State Definitions

Definition	Description
Bridged	<p>This is a G3 PBX private local connection state that is not defined in the CSTA. This state indicates that a call is present at a bridged, simulated bridged, button TEG, or POOL appearance, and the call is neither ringing nor connected at the station. The bridged connection state is reported in the private data of a Snapshot Device Confirmation Event and it has a CSTA null (CS_NULL) state. Since this is the only time G3PD returns CS_NULL, a device with the null state in the Snapshot Device Confirmation Event is bridged.</p> <p>A device with the bridged state can join the call by either manually answering the call or the cstaAnswerCall Service. Once a bridged device is connected to a call, its state becomes connected. After a bridged device becomes connected, it can drop from the call and become bridged again, if there are other endpoints still on the call.</p> <p>NOTE: Manual drop of a bridged line appearance (from the connected state) from a call will not cause a Connection Cleared Event.</p>

Connection State Syntax

```
typedef enum LocalConnectionState_t {
    CS_NONE = -1,
    CS_NULL = 0,
    CS_INITIATE = 1,
    CS_ALERTING = 2,
    CS_CONNECT = 3,
    CS_HOLD = 4,
    CS_QUEUED = 5,
    CS_FAIL = 6
} LocalConnectionState_t;
```

G3 CSTA System Capacity

Table 3-8 provides the system capacity information for the G3 CSTA. These are maximum system capacities. The numbers shown, as well as the server's hardware configuration and the switch configuration, limit a Windows NT or NetWare Telephony Server's capacity. The number of users that can access a Telephony Server is independent of these numbers⁴.

Table 3-8. CSTA System Capacities

Parameter	G3i System Capacity	G3s System Capacity	G3r System Capacity	G3 PBX Driver Capacity	Comments
CTI Links	Eight	Four	Eight	Four	Up to four are supported by one G3PD
CSTA Service Requests per CTI Link	Limited by the following numbers	Limited by the following numbers	Limited by the following numbers	2000 per CTI link	See Note 1.
Objects monitored by cstaMonitorDevice requests	2000 per G3 switch	250 per G3 switch	6000 per G3 switch	Limited by the lesser of switch capacity and link capacity	Maximum number of monitored stations. See Note 2.
Objects monitored by cstaMonitorCallsVia Device requests	170 per G3 switch	50 per G3 switch	460 per G3 switch; 2000 for release G3V3 and later	Limited by switch capacity and link capacity	Maximum number of monitored VDNs and ACD splits allowed. See Note 2.
Simultaneous cstaMonitorDevice monitor requests on one station device	Two, but a G3PD multiplexes client requests into a single association.	Two, but a G3PD multiplexes client requests into a single association.	Two, but a G3PD multiplexes client requests into a single association.	No maximum number but limited by G3PD memory.	Monitor requests can come from the same application or different applications.

4. The number of users that can access the Telephony Server may be limited by the purchase agreement.

Table 3-8. CSTA System Capacities

Parameter	G3i System Capacity	G3s System Capacity	G3r System Capacity	G3 PBX Driver Capacity	Comments
Simultaneous cstaMonitorCallsVia Device monitor requests on one ACD device	One, but a G3PD multiplexes client requests into a single association.	One, but a G3PD multiplexes client requests into a single association.	One, but a G3PD multiplexes client requests into a single association.	No maximum number but limited by G3PD memory	Monitor requests can come from the same application or different applications.
Simultaneous CSTA Clear Connection, Clear Call, and Set Feature Service requests	300 per G3 switch	75 per G3 switch	3000 per G3 switch	Limited by switch capacity and link capacity	See Note 3.
Simultaneous CSTA service requests other than the ones listed in the preceding table cell	2000 per G3 switch	250 per G3 switch	6000 per G3 switch	Limited by the lesser of switch capacity and link capacity	Maximum number of monitored stations See Note 2.
Number of simultaneous call classifications in progress (predictive calls in between the make call request and the switch returning a classification)	40 per G3 switch	40 per G3 switch	400 per G3 switch	N/A	
Number of simultaneous outstanding route requests on a G3 CTI link	127	127	4000 (G3V8 and later)	127 (G3V7 and earlier)	

Table 3-8. CSTA System Capacities

Parameter	G3i System Capacity	G3s System Capacity	G3r System Capacity	G3 PBX Driver Capacity	Comments
Number of devices that can be on a call	Six	Six	Six	Six	See Note 4.
Number of cstaMonitor CallsViaDevice monitored objects that can be involved in a call	One per G3 switch; three for G3V3 or later	One per G3 switch; three for G3V3 or later	One per G3 switch; three for G3V3 or later	A G3PD multiplexes client requests into a single association.	A call can only be actively monitored via one ACD device. See Note 5.
Number of CSTA monitor requests that can be involved in a call	N/A	N/A	N/A	No maximum number but limited by G3PD memory	Each CSTA Event Report of a monitored object will be sent to every monitor request.

The following notes provide additional information pertaining to CSTA system capacity in Table 3-8.

Note 1 This number consists of all Monitored objects, outstanding Call Control Service requests, and outstanding Set Feature Service requests (as well as the outstanding requests of Query Services and Routing Service). The 2000 number is the default number set for each CTI link. This number can be higher or lower depending on the memory configuration of the G3PD DLL/NLM. This number should be configured according to the administration information in the DEFINITY Network Manager's Guide for Windows NT or NetWare Telephony Services.

Note 2 This is not the number of total monitor requests. An object monitored by multiple monitor requests is counted only once. All Call Control Service requests on a station device other than Clear Connection and Clear Call are included in this number. When a station device is monitored, the Call Control Service requests on the device are not counted as additional requests.

- Note 3 This is an estimated number. This number includes all outstanding Clear Connection Service requests, Set Feature Service requests, and Query Service requests.
- Note 4 A call can have a maximum of six parties.
- Note 5 A call may pass through several ACD devices monitored by `cstaMonitorCallsViaDevice` requests, but only one is active (that is, receives event reports for that call) for that call at one time.

Multiple Telephony Server Considerations

Due to the system capacity limitations, care must be taken when using multiple G3PDs (on the same or different Tservers) for the same G3 PBX switch.

- The simultaneous `cstaMonitorDevice` requests on one station device are limited to two per G3 PBX. A maximum of two G3PDs can monitor the same station at the same time.
- The simultaneous `cstaMonitorCallsViaDevice` monitor requests on one ACD device (VDN or ACD split) are limited to one per G3 PBX. A maximum of one G3PD can monitor the same ACD device at a time.
- A call may pass through an ACD device monitored by one G3PD and be redirected to another ACD device monitored by another G3PD. The former will lose the event reports of that call after Diverted Event Report. Similar cases can result when two calls that are monitored by `cstaMonitorCallsViaDevice` requests from different G3PDs are merged (transfer or conference operations or requests) into one.

Multiple CTI Link Considerations

For NetWare, when a G3PD is configured to provide multiple advertised services using multiple links to a G3 PBX, the above G3 CSTA limits above apply the G3PD, not to each advertised service. Thus, for example, for a given call there can be one `cstaMonitorCallsViaDevice` association between the switch and the G3PD. As the table states, however, multiple clients (and these may be clients that have opened streams to different advertised services) may monitor the call using `cstaMonitorCallsViaDevice`.

If a link to a G3 PBX becomes unavailable, all monitors or controls using that link terminate. New monitors or feature requests will be made across any remaining links to the G3 PBX.

During initialization, G3PD advertises for each G3 PBX configured with a link in g3pd.ini even though none of the links to the G3 PBX may be in service. G3PD ceases to advertise when the G3PD DLL/NLM is unloaded. If an application makes an open stream request and there is no link available to the G3 PBX, the application will receive an ACS Universal Failure with code (DRIVER_LINK_UNAVAILABLE).

If all links to a G3 PBX become unavailable, any previously opened streams remain open until the application closes them or the G3PD unloads. The application will not receive a message indicating that there are no links unless the application has used cstaSysStatStart to request system status event reporting.

If a CTI link to a G3 PBX goes down, G3PD sends:

- a CSTA Universal Failure event for each outstanding request (cstaMakeCall(), etc.). An outstanding CSTA request is one that has not yet received a confirmation event. The error code is set to RESOURCE_OUT_OF_SERVICE (34). The client should re-issue the request. If other links are available, the new request will succeed. If no other links are available, the client will continue to receive RESOURCE_OUT_OF_SERVICE (34) and should assume service is unavailable.
- a Monitor End event for any previously established monitor requests. The cause will be EC_NETWORK_NOT_OBTAINABLE (21). The client should re-establish the monitor request. If other links are available, the monitor request will be honored. If no other links are available, the client will receive a CSTA Universal Failure with the error code set to RESOURCE_OUT_OF_SERVICE (34) and should assume service to the switch is unavailable.
- a Route End event for any active Route Select dialogue. The client need do nothing.
- a Route Register Abort event will be sent to the application when all of the CTI links that support the routeRequestEvents for the registered routing device are down. The application could make use of System and Link Status Notification (see Chapter 11, "System Status Service Group") to determine when the link comes back up. If the application wants to continue the routing service after the CTI link is up, it must issue a cstaRouteRegisterReq() to re-establish a routing registration session for the routing device.
- The system status services and events provide G3 private data that inform applications of the status of the multiple links to a G3 PBX. Refer to Chapter 11, "System Status Service Group".

Format and Conventions

This following general format is used to describe each G3 CSTA Service.

Direction:	Direction of the service request or event report across the TSAPI interface: <ul style="list-style-type: none"> ■ Client to Switch client/application to switch/G3PD ■ Switch to Client switch/G3PD to client/application
Function and Confirmation Event:	CSTA service request function and CSTA confirmation event as defined in <i>Telephony Services Application Programming Interface (TSAPI)</i> .
Private Data Function and Private Data Confirmation Event	Private data setup function and private data confirmation event, if any. This function may be called to setup private parameters, if any. This function returns an error, if there is an error in the private parameters. An application should check the return value to make sure that the private data is set up correctly before sending the request to the G3PD.
Service Parameters:	List of parameters for this service request. Common ACS parameters such as acsHandle, invokeID, and privateData are not shown.
Private Parameters:	List of parameters that can be specified in private data for this service request.
Ack Parameters:	List of parameters in the confirmation event for the positive acknowledgment from the server. Common ACS parameters such as acsHandle, eventClass, eventType, and privateData are not shown.
Ack Private Parameters:	List of parameters in the private data of the confirmation event for the positive acknowledgment from the server.
Nak Parameter: <i>universalFailure</i>	If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain one of the error values described in the “CSTAUniversalFailureConfEvent” section in this chapter
Functional Description	Detailed description of the telephony function that this CSTA Service provides in a G3 CSTA environment.
Service Parameters:	
<i>parameter</i>	Detailed information for each parameter in the service request. A noData indicator means that it requires no additional parameters other than the common ACS parameters. The mandatory/optional attribute of a parameter is defined as follows:

<i>mandatory</i>	[mandatory] This parameter is mandatory as defined in Standard ECMA-179. It must be present in the service request. If not, the service request will be denied with OBJECT_NOT_KNOWN.
<i>mandatoryPartially</i>	[mandatory - partially supported] This parameter is mandatory as defined in Standard ECMA-179. However, G3 CSTA can only support part of the parameter due to the G3 PBX feature limitations. The G3PD sets a G3 default value for the portion not supported.
<i>mandatoryNotSupt</i>	[mandatory - not supported] This parameter is mandatory as defined in Standard ECMA-179. However, G3 CSTA does not support this parameter due to the G3 PBX feature limitations. "Not supported" means that whether the application passes it or not, the value specified will be ignored and a default value will be assigned. If this is a parameter (for example, event report parameter) returned from the switch, the G3PD sets a G3 default value for this parameter.
<i>optional</i>	[optional] This parameter is optional as defined in Standard ECMA-179. It may or may not be present in the service request. If not, the G3PD sets a G3 default value.
<i>optionalSupported</i>	[optional - supported] This parameter is optional as defined in Standard ECMA-179, but it is always supported.
<i>optionalPartially</i>	[optional - partially supported] This parameter is optional as defined in Standard ECMA-179. However, G3 CSTA Services can only support part of the parameter due to the G3 PBX feature limitations. The part that is not supported will be ignored, if it is present.
<i>optionalNotSupport</i>	[optional - not supported] This parameter is optional as defined in Standard ECMA-179, but it is not supported by G3 CSTA Services. "Not supported" means that whether the application passes it or not, the value specified will be ignored and the G3PD will assign a G3 default value.
<i>optionalLimitedSupt</i>	[optional - limited support] This parameter is optional as defined in Standard ECMA-179, but it is not fully supported by G3 CSTA Services.

⇒ NOTE:

An application must understand the limitations of this parameter in order to use the information correctly. The limitations are described in the Detailed Information section associated with each service.

Private Service Parameters:

parameter Detailed information for each private parameter in the service request.

The mandatory/optional attribute of a parameter is defined as follows:

- mandatory*** [mandatory] This parameter is mandatory for the specific service. It must be present in the private data of the request. If not, the service request will be denied by the G3PD with OBJECT_NOT_KNOWN.
- optional*** [optional] This parameter is optional for the specific service. It may or may not be present in the private data. If not, G3PD will assign a G3 default value.
- optionalNotSupported*** [optional — not supported] This parameter is optional for the specific service. This parameter is reserved for future use. It is ignored for the current implementation.

Ack Parameters:

- parameter** Detailed information for each parameter in the service confirmation event. A noData indicator means that the G3PD sends no additional parameters other than the confirmation event itself along with the common ACS parameters.

Ack Private Parameters:

- parameter*** Detailed information for each parameter in the private data of the service confirmation event.

Nak Parameter:

- universalFailure*** If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain one of the error values described in the “CSTAUniversalFailureConfEvent” section in this chapter.

Detailed Information: Detailed information about switch operations, feature interactions, restrictions, and special rules.

Syntax: C-declarations of the TSAPI function and the confirmation event for this service.

Private Parameter Syntax: C-declarations of the private parameters and the set up functions and of the private parameters in the confirmation event for this service.

Example: Programming examples are given for some of the services and events.

Common ACS Parameter Syntax

```
typedef unsigned longInvokeID_t;

typedef unsigned shortACSHandle_t;

typedef unsigned shortEventClass_t;

typedef unsigned shortEventType_t;

// defines for ACS event classes

#define ACSREQUEST 0
#define ACSUNSOLICITED 1
#define ACSCONFIRMATION 2

// defines for CSTA event classes

#define CSTAREQUEST 3
#define CSTAUNSOLICITED 4
#define CSTACONFIRMATION 5
#define CSTAEVENTREPORT 6
```


CSTAUniversalFailureConfEvent

The CSTA universal failure confirmation event provides a generic negative response from the server/switch for a previously requested service. The CSTAUniversalFailureConfEvent will be sent in place of any confirmation event described in each service function description when the requested function fails. The confirmation events defined for each service function are only sent when that function completes successfully.

Here is a list of most commonly used CSTA errors returned by G3 CSTA Services in the CSTAUniversalFailureConfEvent for a negative acknowledgment of this service.

⇒ NOTE:

This list does not include those error codes that are returned by the Tserver EXE/NLM. Those error codes (for example, security violation detected by Tserver) returned by the Tserver EXE/NLM are documented in Telephony Services Application Programming Interface (TSAPI).

This list does not include all possible errors. An application program should be able to handle any CSTA error defined in the CSTAUniversalFailure_t. Failure to do so may cause the application program to fail.

The following common errors apply to every CSTA Service supported by G3 CSTA Services. They will not be repeated for each service description.

Table 3-9. Common CSTA Service Errors

<i>universalFailure</i>	<ul style="list-style-type: none"> ■ If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter may contain one of the following error values: ■ GENERIC_OPERATION (1) The CTI protocol has been violated or the service invoked is not consistent with a CTI application association. Report this error to — see “Customer Support” on page 1-6. ■ REQUEST_INCOMPATIBLE_WITH_OBJECT (2) The service request does not correspond to a CTI application association. Report this error — see “Customer Support” on page 1-6. ■ VALUE_OUT_OF_RANGE (3) The G3 PBX switch detects that a required parameter is missing in the request or an out-of-range value has been specified.
--------------------------------	---

- **OBJECT_NOT_KNOWN (4)** The G3PD detects that a required parameter is missing in the request. For example, the deviceID of a connectionID is not specified in a service request.
- **INVALID_FEATURE (15)** The G3PD detects a CSTA Service request that is not supported by the G3 PBX switch.
- **GENERIC_SYSTEM_RESOURCE_AVAILABILITY (31)** The request cannot be executed due to a lack of available switch resources.
- **RESOURCE_OUT_OF_SERVICE (34)** An application can receive this error code when a single CSTA Service request is ending abnormally due to protocol error.
- **NETWORK_BUSY (35)** The PBX switch is not accepting the request at this time because of processor overload. The application may wish to retry the request but should not do so immediately.
- **OUTSTANDING_REQUEST_LIMIT_EXCEEDED (44)** The given request cannot be processed due to the system resource limit on the device.
- **GENERIC_UNSPECIFIED_REJECTION (70)** This is a G3 PBX Driver internal error, but it cannot be any more specific. A system administrator may find more detailed information about this error in the G3PD error logs. Report this error — see “Customer Support” on page 1-6.
- **GENERIC_OPERATION_REJECTION (71)** This is a G3 PBX Driver internal error, but not a defined error. A system administrator should check the G3PD error logs for more detailed information about this error. Report this error — see “Customer Support” on page 1-6.
- **DUPLICATE_INVOCATION_REJECTION (72)** The G3PD detects that the invokeID in the service request is being used by another outstanding service request. This service request is rejected. The outstanding service request with the same invokeID is still valid.
- **UNRECOGNIZED_OPERATION_REJECTION (73)** The G3PD detects that the service request from a client application is not defined in the API. A CSTA request with a 0 or negative invokeID will receive this error.

- RESOURCE_LIMITATION_REJECTION (75) The G3PD detects that it lacks internal resources such as the memory or data records to process a service request. A system administrator should check the G3PD error logs for more detailed information about this error. This failure may reflect a temporary situation. The application should retry the request.
- ACS_HANDLE_TERMINATION_REJECTION (76) The G3PD detects that an acsOpenStream session is terminating. The G3PD sends this error for every outstanding CSTA request of this ACS Handle. If the session is not closed in an orderly fashion, the application may not receive this error. For example, a user may power off the PC before the application issues an acsCloseStream request and waits for the confirmation event. In this case, the acsCloseStream is issued by the Tserver on behalf of the application and there is no application to receive this error. If an application issues an acsCloseStream request and waits for its confirmation event, the application will receive this error for every outstanding request.

- **SERVICE_TERMINATION_REJECTION (77)** The G3PD detects that it cannot provide the service due to the failure or shutting down of the communication link between the Telephony Server and the G3 PBX. The G3PD sends this error for every outstanding CSTA request for every ACS Handle affected. Although the link is down or the switch is out of service, the G3PD remains loaded and advertised. When the G3PD is in this state, all CSTA Service requests from a client will receive a negative acknowledgment with this unique error code.
- **REQUEST_TIMEOUT_REJECTION (78)** The G3PD did not receive the response of a service request sent to the G3 PBX more than 20 seconds ago. The timer of the request has expired. The request is canceled and negatively acknowledged with this unique error code. When this occurs, the communication link between the Telephony Server and the G3 switch may be congested. This can happen when the PBX and/or the Tserver exceeds their capacity.
- **REQUESTS_ON_DEVICE_EXCEEDED_REJECTION (79)** For a device, the G3 PBX processes one service request at a time. The G3PD queues CSTA requests for a device. Only a limited number of CSTA requests can be queued on a device. This number is defined in the `MAX_REQS_QUEUED_PER_DEV` parameter in the `g3pd.ini`¹ file. If this number is exceeded, the incoming client request is negatively acknowledged with this unique error code. Usually an application sends one request and waits for its completion before it makes another request. The `MAX_REQS_QUEUED_PER_DEV` parameter has no effect on this kind of operation. Situations of sending a sequence of requests without waiting for their completion are rare. However, if this is the case, the `MAX_REQS_QUEUED_PER_DEV` parameter should be set to a proper value. The default value for `MAX_REQS_QUEUED_PER_DEV` is 4.

1. See the DEFINITY ECS Network Manager's Guide for your product.

Syntax

The following structure shows only the relevant portions of the unions for this message:

```
typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass;
    EventType_t eventType;
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;

    union
    {
        struct
        {
            InvokeID_t invokeID;
            union
            {
                CSTAUniversalFailureConfEvent_t universalFailure;
            } u;
        } cstaConfirmation;
    } event;
} CSTAEvent_t;

typedef struct CSTAUniversalFailureConfEvent_t {
    CSTAUniversalFailure_t error;
} CSTAUniversalFailureConfEvent_t;
```

The universalFailure error codes are listed below:

```
typedef enum CSTAUniversalFailure_t {
    GENERIC_UNSPECIFIED = 0,
    GENERIC_OPERATION = 1,
    REQUEST_INCOMPATIBLE_WITH_OBJECT = 2,
    VALUE_OUT_OF_RANGE = 3,
    OBJECT_NOT_KNOWN = 4,
    INVALID_CALLING_DEVICE = 5,
    INVALID_CALLED_DEVICE = 6,
    INVALID_FORWARDING_DESTINATION = 7,
    PRIVILEGE_VIOLATION_ON_SPECIFIED_DEVICE = 8,
    PRIVILEGE_VIOLATION_ON_CALLED_DEVICE = 9,
    PRIVILEGE_VIOLATION_ON_CALLING_DEVICE = 10,
    INVALID_CSTA_CALL_IDENTIFIER = 11,
    INVALID_CSTA_DEVICE_IDENTIFIER = 12,
    INVALID_CSTA_CONNECTION_IDENTIFIER = 13,
    INVALID_DESTINATION = 14,
    INVALID_FEATURE = 15,
}
```

Syntax (Continued)

```
INVALID_ALLOCATION_STATE = 16,
INVALID_CROSS_REF_ID = 17,
INVALID_OBJECT_TYPE = 18,
SECURITY_VIOLATION = 19,

GENERIC_STATE_INCOMPATIBILITY = 21,
INVALID_OBJECT_STATE = 22,
INVALID_CONNECTION_ID_FOR_ACTIVE_CALL = 23,
NO_ACTIVE_CALL = 24,
NO_HELD_CALL = 25,
NO_CALL_TO_CLEAR = 26,
NO_CONNECTION_TO_CLEAR = 27,
NO_CALL_TO_ANSWER = 28,
NO_CALL_TO_COMPLETE = 29,
GENERIC_SYSTEM_RESOURCE_AVAILABILITY = 31,
SERVICE_BUSY = 32,
RESOURCE_BUSY = 33,
RESOURCE_OUT_OF_SERVICE = 34,
NETWORK_BUSY = 35,
NETWORK_OUT_OF_SERVICE = 36,
OVERALL_MONITOR_LIMIT_EXCEEDED = 37,
CONFERENCE_MEMBER_LIMIT_EXCEEDED = 38,
GENERIC_SUBSCRIBED_RESOURCE_AVAILABILITY = 41,
OBJECT_MONITOR_LIMIT_EXCEEDED = 42,
EXTERNAL_TRUNK_LIMIT_EXCEEDED = 43,
OUTSTANDING_REQUEST_LIMIT_EXCEEDED = 44,
GENERIC_PERFORMANCE_MANAGEMENT = 51,
PERFORMANCE_LIMIT_EXCEEDED = 52,
SEQUENCE_NUMBER_VIOLATED = 61,
TIME_STAMP_VIOLATED = 62,
PAC_VIOLATED = 63,
SEAL_VIOLATED = 64,           // The above errors are detected
                               // either by the switch or by
                               // the G3PD.

                               // The following rejections are
                               // generated by the G3PD, not by
// the switch.
GENERIC_UNSPECIFIED_REJECTION = 70,
GENERIC_OPERATION_REJECTION = 71,
DUPLICATE_INVOCATION_REJECTION = 72,
UNRECOGNIZED_OPERATION_REJECTION = 73,
MISTYPED_ARGUMENT_REJECTION = 74,
RESOURCE_LIMITATION_REJECTION = 75,
ACS_HANDLE_TERMINATION_REJECTION = 76,
SERVICE_TERMINATION_REJECTION = 77,
REQUEST_TIMEOUT_REJECTION = 78,
REQUESTS_ON_DEVICE_EXCEEDED_REJECTION = 79
} CSTAUniversalFailure_t;
```

ACUniversalFailureConfEvent

Error values in this category indicate that the G3PD detected an ACS-related error. This type includes one of the following specific error values:

Table 3-10. ACS-Related Errors

- DRIVER_DUPLICATE_ACSHANDLE (1000) The acsHandle given for an ACS Stream request is already in use for a session. The already open session with the acsHandle is remains open.
- DRIVER_INVALID_ACS_REQUEST (1001) The ACS message contains an invalid or unknown request. The request is rejected.
- DRIVER_ACS_HANDLE_REJECTION (1002) The request is rejected because a CSTA request was issued with no prior acsOpenStream request or the acsHandle given for an acsOpenStream request is 0 or negative.
- DRIVER_INVALID_CLASS_REJECTION (1003) The driver received a message containing an invalid or unknown message class. The request is rejected.
- DRIVER_GENERIC_REJECTION (1004) The driver detected an invalid message for something other than message type or message class. This is an internal error and should be reported — see “Customer Support” on page 1-6.

- DRIVER_RESOURCE_LIMITATION (1005) The driver did not have adequate resources (that is memory, etc.) to complete the requested operation. This is an internal error and should be reported — see “Customer Support” on page 1-6.
- DRIVER_ACSHANDLE_TERMINATION (1006) Due to problems with the link to the PBX, the driver has found it necessary to terminate the session with the given acsHandle. The session will be closed, and all outstanding requests will terminate.
- DRIVER_LINK_UNAVAILABLE (1007) The driver was unable to open the new session because no link was available to the PBX. The link may have been placed in the BLOCKED state, it may have been taken off line, or some other link failure may have occurred. When the link is in this state, G3PD remains loaded and advertised and sends this error for every new acsOpenStream request until the link becomes available again. A previously opened session will remain open when the link is in this state. It will receive no specific notification about the link status unless it attempts a CSTA request. In this state, a CSTA request will receive a CSTA Universal Failure with the error SERVICE_TERMINATION_REQUEST.

Overview

The services in this group enable a telephony client application to control a call or connection on the G3 switch. These services are typically used for placing calls from a device and controlling any connection on a single call as the call moves through the G3 switch.

In the following subsections each Call Control Service supported in the Windows NT or NetWare Telephony Services product is illustrated using the conventions below:

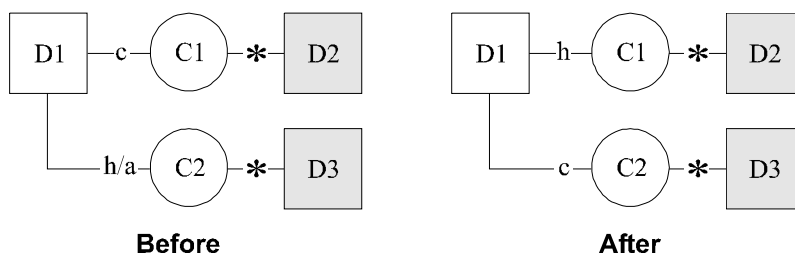
- Boxes represent devices and D1, D2, and D3 represent deviceIDs.
- Circles represent calls and C1, C2, and C3 represent callIDs.
- Lines represent connections between a call and a device; and C1-D1, C1-D2, C2- D3, etc., represent connectionIDs.
- The absence of a line is equivalent to a connection in the Null connection state.
- Labels in boxes and circles represent call and device instances.
- Labels on lines represent a connection state using the following key:
 - a = Alerting
 - c = Connected
 - f = Failed
 - h = Held
 - i = Initiated
 - q = Queued

- a/h = Alerting or Held
- * = Unspecified

- Grayed boxes represent devices in a call unaffected by the service or event report.
- White boxes and circles represent devices and calls affected by the service or event report.
- The parameters for the function call of the service are indicated in bold italic font.

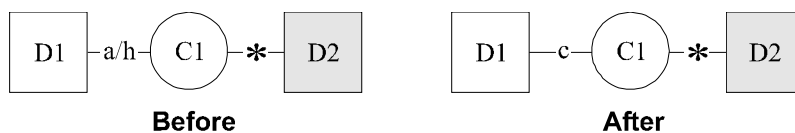
Alternate Call Service

The Alternate Call Service provides a compound action of the Hold Call Service followed by Retrieve Call Service/Answer Call. The Alternate Call Service places an existing activeCall (C1- D1) at a device to another device (D2) on hold and, in a combined action, retrieves/establishes a held/delivered otherCall (C2-D1) between the same device D1 and another device (D3) as the active call. Device D2 can be considered as being automatically placed on hold immediately prior to the retrieval/establishment of the held/alerting call to device D3. A successful service request will cause the held/alerting call to be swapped with the active call.



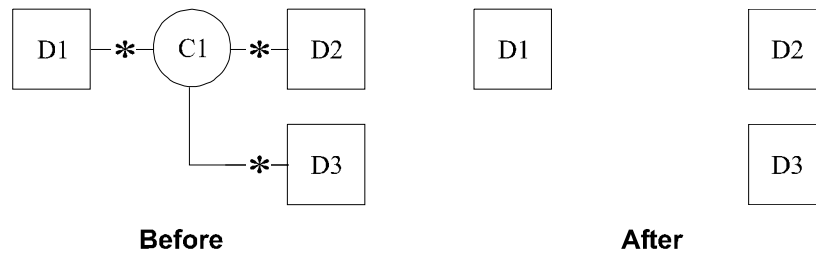
Answer Call Service

The Answer Call Service is used to answer an incoming call (C1) that is alerting a device (D1) with the connection alertingCall (C1-D1). This service is typically used with telephones that have attached speakerphone units to establish the call in a hands-free operation. The Answer Call Service can also be used to retrieve a call (C1) that is held by a device (D1) with the connection alertingCall (C1-D1).



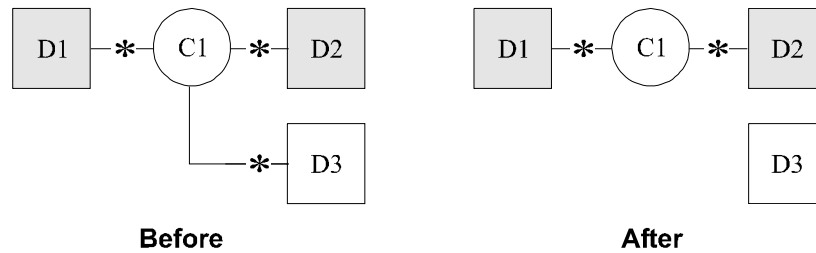
Clear Call Service

This service will cause each device associated with a call (C1) to be released and the connectionIDs (and their components) to be freed.



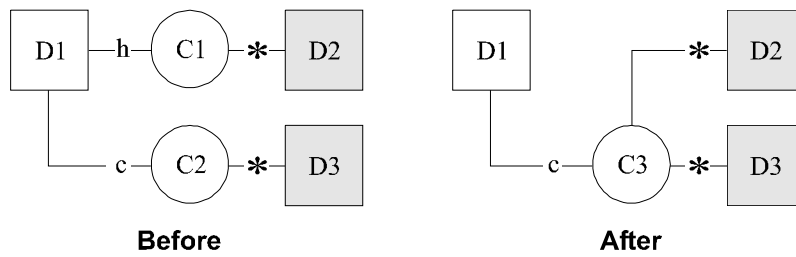
Clear Connection Service

This service releases the specified connection, call (C1-D3), and its connectionID instance from the designated call (C1). The result is as if the device had hung up on the call. The phone does not have to be physically returned to the switchhook, which may result in silence, dial tone, or some other condition. Generally, if only two connections are in the call, the effect of `cstaClearConnection` is the same as `cstaClearCall`. Note that it is likely that the call (C1) is not cleared by this service if it is some type of conference.



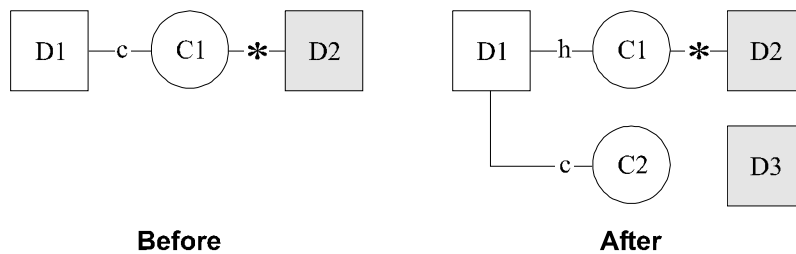
Conference Call Service

This service provides the conference of an existing heldCall (C1-D1) and another activeCall (C2-D1) at the same device. The two calls are merged into a single call (C3) and the two connections (C1-D1, C2-D1) at the conferencing device (D1) are resolved into a single connection, newCall (C3-D1), in the Connected state.



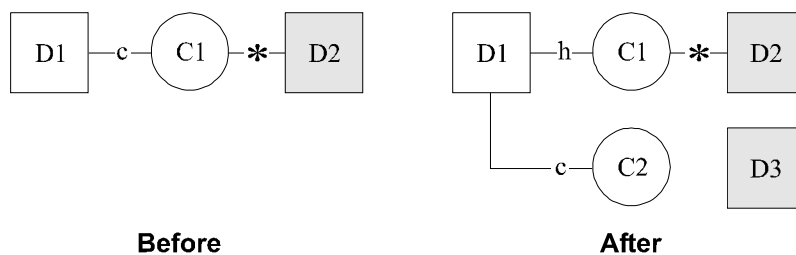
Consultation Call Service

The Consultation Call Service will provide the compound action of the Hold Call Service followed by Make Call Service. This service places an active activeCall (C1-D1) at a device (D1) on hold and initiates a new call from the same device D1 to another calledDevice (D3). The client is returned with the connection newCall (C2-D1).



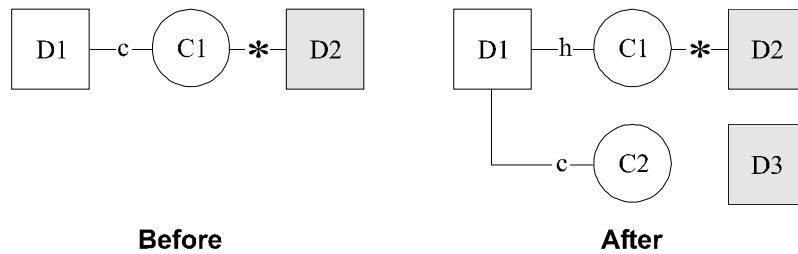
Consultation Direct-Agent Call Service

The Consultation Direct-Agent Call Service will provide the compound action of the Hold Call Service followed by Make Direct-Agent Call Service. This service places an active activeCall (C1-D1) at a device (D1) on hold and initiates a new direct-agent call from the same device D1 to another calledDevice (D3). The client is returned with the connection newCall (C2-D1).



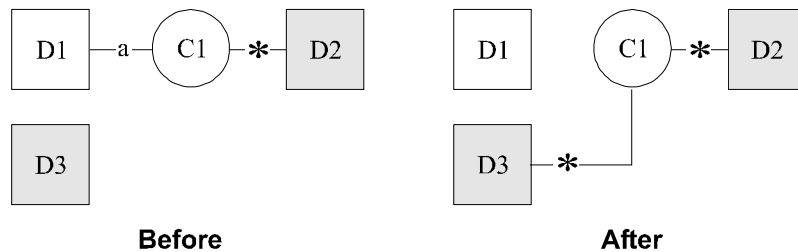
Consultation Supervisor-Assist Call Service

The Consultation Supervisor-Assist Call Service will provide the compound action of the Hold Call Service followed by Make Supervisor-Assist Call Service. This service places an active activeCall (C1-D1) at a device (D1) on hold and initiates a new supervisor-assist call from the same device D1 to another calledDevice (D3). The client is returned with the connection newCall (C2-D1).



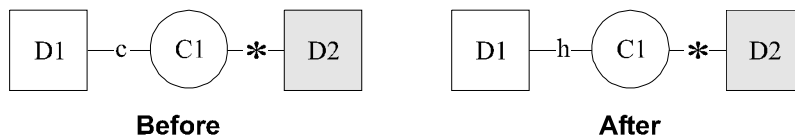
Deflect Call Service

The Deflect Call Service redirects an alerting call (C1) at a device (D1) with the connection deflectCall to a new destination, either on-PBX or off-PBX.



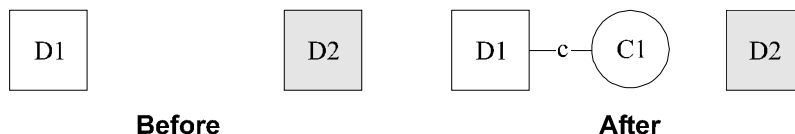
Hold Call Service

The Hold Call Service places a call (C1) at a device (D1) with the connection activeCall (C1-D1) on hold. The effect is as if the specified party depressed the hold button on the device or flashed the switchhook to locally place the call on hold. The call is usually in the active state. This service maintains a relationship between the holding device (D1) and the held call (C1) that lasts until the call is retrieved from the hold status or until the call is cleared.



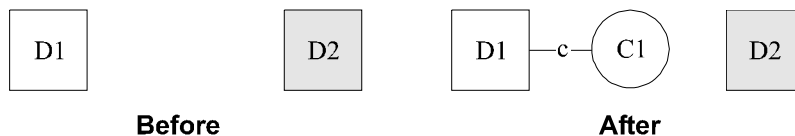
Make Call Service

The Make Call Service originates a call between two devices designated by the application. When the service is initiated, the callingDevice (D1) is prompted (if necessary), and when that device acknowledges, a call to the calledDevice (D2) is originated. A call is established as if D1 had called D2, and the client is returned with the connection newCall (C1-D1).



Make Direct-Agent Call Service

The Make Direct-Agent Call Service originates a call between two devices: a user station and an ACD agent logged into a specified split. When the service is initiated, the callingDevice (D1) is prompted (if necessary), and when that device acknowledges, a call to the calledDevice (D2) is originated. A call is established as if D1 had called D2, and the client is returned with the connection newCall (C1-D1).



The Make Direct Agent Call Service should be used only in the following two situations:

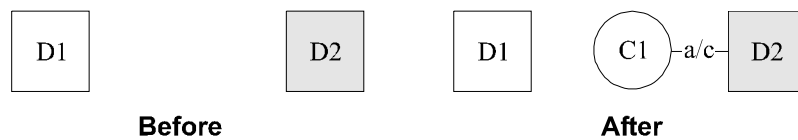
- Direct Agent Calls in a non-EAS environment
- Direct Agent Calls in an EAS environment only when it is required to ensure that these calls against a skill other than that skill specified for these measurements on the DEFINITY PBX for that agent.

Preferably in an EAS environment, Direct Agent Calls can be made using the Make Call service and specifying an Agent login-ID as the destination device. In

this case Direct Agent Calls will be measured against the skill specified or those measurements on the DEFINITY PBX for that agent.

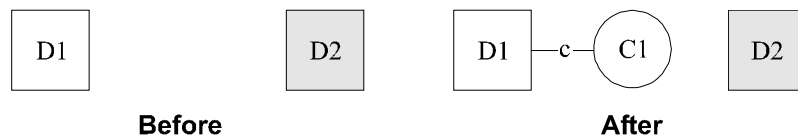
Make Predictive Call Service

The Make Predictive Call Service originates a Switch-Classified call between two devices. The service attempts to create a new call and establish a connection with the calledDevice (D2) first. The client is returned with the connection newCall (C1-D2).



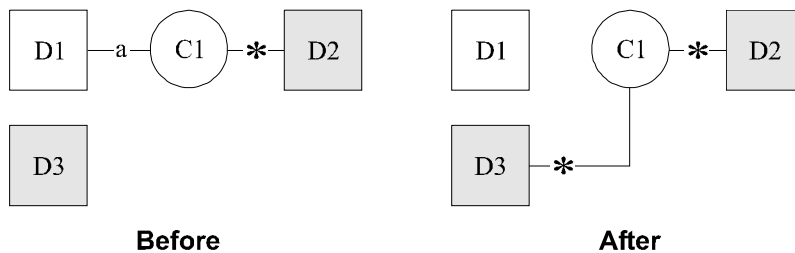
Make Supervisor-Assist Call Service

The Make Supervisor-Assist Call Service originates a supervisor-assist call between two devices: an ACD agent station and another station (typically a supervisor). When the service is initiated, the callingDevice (D1) is prompted (if necessary), and when that device acknowledges, a call to the calledDevice (D2) is originated. A call is established as if D1 had called D2, and the client is returned with the connection newCall (C1-D1).



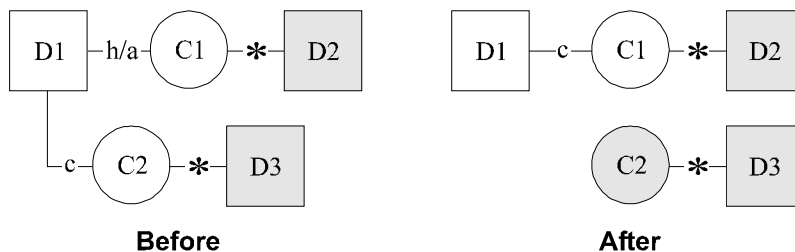
Pickup Call Service

The Pickup Call Service takes an alerting call (C1) at a device (D1) with the connection deflectCall to another on-PBX device.



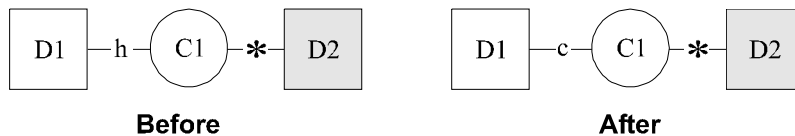
Reconnect Call Service

The Reconnect Call Service allows a client to disconnect an existing connection activeCall (C2- D1) from a call and then retrieve/establish a previously held/delivered connection heldCall (C1- D1).



Retrieve Call Service

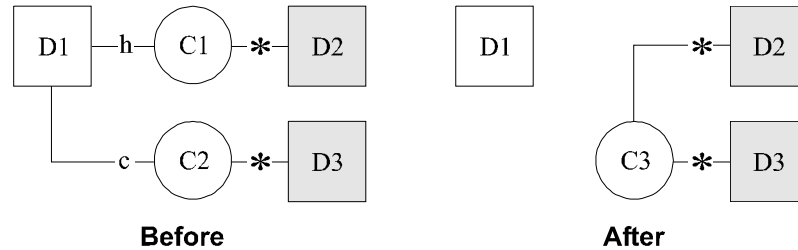
The service restores a held connection heldCall (C1-D1) to the Connected state (active).



Transfer Call Service

This service provides the transfer of a heldCall (C1-D1) with an activeCall (C2-D1) at the same device (D1). The transfer service merges two calls (C1, C2) with connections (C3-D2, C3-D3) at a single common device (D1) into one call (C3). Also, both of the connections to the common device become Null and their connectionIDs are released. When the transfer completes, the common device

(D1) is released from the calls (C1, C2). A callID, newCall (C3) that specifies the resulting new call for the transferred call is provided.



Alternate Call Service

Direction: Client to Switch
Function: *cstaAlternateCall ()*
Confirmation Event: *CSTAAAlternateCallConfEvent*
Service Parameters: *activeCall, otherCall*
Ack Parameters: *noData*
Nak Parameter: *universalFailure*

Functional Description:

The Alternate Call Service allows a client to put an existing active call (*activeCall*) on hold and then answer an alerting (or bridged) call or retrieve a previously held call (*otherCall*) at the same station. It provides the compound action of the Hold Call Service followed by an Answer Call Service or a Retrieve Call Service.

The Alternate Call Service request is acknowledged (Ack) by the switch if the switch is able to put the *activeCall* on hold and

- connect the specified alerting *otherCall* either by forcing the station off-hook (turning the speakerphone on) or waiting up to five seconds for the user to go off- hook, or
- retrieve the specified held *otherCall*.

The request is negatively acknowledged if the switch:

- fails to put *activeCall* on hold (for example, call is in alerting state),
- fails to connect the alerting *otherCall* (for example, call dropped), or
- fails to retrieve the held *otherCall*.

If the request is negatively acknowledged, the G3PD will attempt to put the *activeCall* to its original state, if the original state is known by the G3PD before the service request. If the original state is unknown, there is no recovery for the *activeCall*'s original state.

Service Parameters:

activeCall [mandatory] A valid connection identifier that indicates the callID and the station extension (STATIC_ID). The deviceID in activeCall must contain the station extension of the controlling device. The local connection state of the call can be either active or held.

otherCall [mandatory] A valid connection identifier that indicates the callID and the station extension (STATIC_ID). The deviceID in otherCall must contain the station extension of the controlling device. The local connection state of the call can be either alerting, bridged, or held.

Ack Parameter:

noData None for this service.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- INVALID_CSTA_DEVICE_IDENTIFIER (12) An invalid device identifier or extension is specified in activeCall or otherCall.
- INVALID_CSTA_CONNECTION_IDENTIFIER (13) An incorrect callID, a incorrect deviceID, or dynamic device ID type is specified in activeCall or otherCall.
- GENERIC_STATE_INCOMPATIBILITY (21) The otherCall station user did not go off-hook within five seconds and is not capable of being forced off-hook.
- INVALID_OBJECT_STATE (22) The otherCall is not in the alerting, connected, held, or bridged state.
- INVALID_CONNECTION_ID_FOR_ACTIVE_CALL (23) The controlling deviceID in activeCall and otherCall is different.
- NO_ACTIVE_CALL (24) The activeCall to be put on hold is not currently active (in alerting state, for example) so it cannot be put on hold.
- NO_CALL_TO_ANSWER (28) The otherCall was redirected to coverage within the five- second interval.

- **GENERIC_SYSTEM_RESOURCE_AVAILABILITY (31)**
The client attempted to add a seventh party (otherCall) to a call with six active parties.
- **RESOURCE_BUSY (33)** User at the otherCall station is busy on a call or there is no idle appearance available. It is also possible that the switch is busy with another CSTA request. This can happen when two G3PDs are issuing requests (for example, Hold Call, Retrieve Call, Clear Connection, etc.) to the same device.
- **OUTSTANDING_REQUEST_LIMIT_EXCEEDED (44)**
The client attempted to put a third party (activeCall) on hold (two parties are on hold already) on an analog station.
- **MISTYPED_ARGUMENT_REJECTION (74)**
DYNAMIC_ID is specified in activeCall or otherCall.

Detailed Information:

See “Detailed Information:” in the “Answer Call Service” section and “Detailed Information:” in the “Hold Call Service” section in this chapter.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaAlternateCall() - Service Request

RetCode_t      cstaAlternateCall (
    ACSHandle_t   acsHandle,
    InvokeID_t    invokeID,
    ConnectionID_t *activeCall,    // devIDType= STATIC_ID
    ConnectionID_t *otherCall,    // devIDType= STATIC_ID
    PrivateData_t *privateData);

// CSTAAlternateCallConfEvent - Service Response

typedef struct
{
    ACSHandle_t   acsHandle;
    EventClass_t  eventClass; // CSTACONFIRMATION
    EventType_t   eventType;  // CSTA_ALTERNATE_CALL_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t  eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
            union
            {
                CSTAAlternateCallConfEvent_t  alternateCall;
            } u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAAlternateCallConfEvent_t {
    Nulltype      null;
} CSTAAlternateCallConfEvent_t;
```

Answer Call Service

Direction: Client to Switch
Function: *cstaAnswerCall ()*
Confirmation Event: *CSTAAnswerCallConfEvent*
Service Parameters: *alertingCall*
Ack Parameters: *noData*
Nak Parameter: *universalFailure*

Functional Description:

The Answer Call Service allows a client application to request on behalf of a station user the ability to answer a ringing or bridged call (*alertingCall*) present at a station. Answering a ringing or bridged call means to connect a call by forcing the station off-hook if the user is on-hook, or cutting the call through to the head or handset if the user is off-hook (listening to dial tone or being in the off-hook idle state). The effect is as if the station user selected the call appearance of the alerting or bridged call and went off-hook.

The *deviceID* in *alertingCall* must contain the station extension of the endpoint to be answered on the call. A Delivered Event Report must have been received by the application prior to this request.

The Answer Call Service can be used to answer a call present at any station type (for example, analog, DCP, hybrid, and BRI).

The Answer Call Service request is acknowledged (Ack) by the switch if the switch is able to connect the specified call either by forcing the station off-hook (turning on the speakerphone) or waiting up to five seconds for the user to go off-hook. Answering a call that is already connected or in the held state will result in a positive acknowledgment and, if the call was held, the call becomes connected.

Service Parameters:

alertingCall [mandatory] A valid connection identifier that indicates the callID and the station extension (STATIC_ID).

Ack Parameter:

noData None for this service.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- INVALID_CSTA_DEVICE_IDENTIFIER (12) An invalid device identifier or extension is specified in alertingCall.
- INVALID_CSTA_CONNECTION_IDENTIFIER (13) An incorrect callID or an incorrect deviceID is specified.
- GENERIC_STATE_INCOMPATIBILITY (21) The station user did not go off-hook within five seconds and is not capable of being forced off-hook.
- INVALID_OBJECT_STATE (22) The specified connection at the station is not in the alerting, connected, held, or bridged state.
- NO_CALL_TO_ANSWER (28) The call was redirected to coverage within the five-second interval.
- GENERIC_SYSTEM_RESOURCE_AVAILABILITY (31) The client attempted to add a seventh party to a call with six active parties.
- RESOURCE_BUSY (33) The user at the station is busy on a call or there is no idle appearance available.
- MISTYPED_ARGUMENT_REJECTION (74) DYNAMIC_ID is specified in alertingCall.

Detailed Information:

- Multifunction Station Operation — For a multifunction station user, this service will be successful in the following situations:
 - The user’s state is being alerted on-hook. For example, the user can either be forced off-hook or is manually taken off-hook within five seconds of the request. The switch will select the ringing call appearance.

- The user is off-hook idle. The switch will select the alerting call appearance and answer the call.
- The user is off-hook listening to dial tone. The switch will drop the dial tone call appearance and answer the alerting call on the alerting call appearance.

A held call will be answered (retrieved) on the held call appearance, provided that the user is not busy on another call. This service is not recommended to retrieve a held call. The `cstaRetrieveCall` Service should be used instead.

A bridged call will be answered on the bridged call appearance, provided that the user is not busy on another call, or the exclusion feature is not active for the call.

An ACB, PCOL, or TEG call will be answered on a free call appearance, provided that the user is not busy on another call.

If the station is active on a call (talking), listening to reorder/intercept tone, or does not have an idle call appearance (for ACB, ICOM, PCOL, or TEG calls) at the time the switch receives the Answer Call Service request, the request will be denied.

- **Analog Station Operation** For an analog station user, the service will be successful only under the following circumstances:
 - The user is being alerted on-hook (and is manually taken off-hook within five seconds).
 - The user is off-hook idle (or listening to dial tone) with a call waiting. The switch will drop the dial tone (if any) and answer the call waiting call.
 - The user is off-hook idle (or listening to dial tone) with a held call (soft or hard). The switch will drop the dial tone (if any) and answer the specified held call (there could be two held calls at the set, one soft-held and one hard-held).

An analog station may only have one or two held calls when invoking the Answer Call Service on a call. If there are two held calls, one is soft-held, the other hard-held. Answer Call Service on any held call (in the absence of another held call and with an off-hook station) will reset the switch-hook flash counter to zero, as if the user had manually gone on-hook and answered the alerting/held call. Answer Call Service on a hard-held call (in the presence of another, soft-held call and with an off-hook station) will leave the switch-hook flash counter unchanged. Thus, the user may use subsequent switch-hook flashes to effect a conference operation between the previously soft-held call and the active call (reconnected from the hard-held call). Answer Call Service on a hard-held call in the presence of another soft-held call and with the station on-hook will be denied. This is consistent with manual operation because when the user goes on-hook with two held calls, one soft-held and one hard-held, the user is again alerted, goes off-hook, and the soft-held call is retrieved.

If the station is active on a call (talking) or listening to reorder/intercept tone at the time the Answer Call Service is requested, the request will be denied (RESOURCE_BUSY).

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaAnswerCall() - Service Request

RetCode_t      cstaAnswerCall (
    ACSHandle_t   acsHandle,
    InvokeID_t    invokeID,
    ConnectionID_t *alertingCall, // devIDType= STATIC_ID
    PrivateData_t *privateData);

// CSTAAnswerCallConfEvent - Service Response

typedef struct
{
    ACSHandle_t   acsHandle;
    EventClass_t  eventClass; // CSTACONFIRMATION
    EventType_t  eventType; // CSTA_ANSWER_CALL_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t  eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
            union
            {
                CSTAAnswerCallConfEvent_t  answerCall;
            } u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAAnswerCallConfEvent_t {
    Nulltype      null;
} CSTAAnswerCallConfEvent_t;
```

Clear Call Service

Direction: Client to Switch
Function: *cstaClearCall ()*
Confirmation Event: *CSTAClearCallConfEvent*
Service Parameters: *call*
Ack Parameters: *noData*
Nak Parameter: *universalFailure*

Functional Description:

The Clear Call Service disconnects all connections from the specified call and terminates the call itself. All connection identifiers previously associated with the call are no longer valid.

Service Parameters:

call [mandatory] A valid connection identifier that indicates the call to be cleared. The deviceID of call is optional. If it is specified, it is ignored.

Ack Parameter:

noData None for this service.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a *CSTAUniversalFailureConfEvent*. The error parameter in this event may contain the following error values, or one of the error values described in the “*CSTAUniversalFailureConfEvent*” section in Chapter 3:

- **NO_ACTIVE_CALL (24)** The callID of the connectionID specified in the request is invalid.

Detailed Information:

- Switch operation — After a successful Clear Call Service request:
 - Every station dropped will be in the off-hook idle state.
 - Any lamps associated with the call are off.
 - Displays are cleared.
 - Auto-answer analog stations do not receive dial tone.
 - Manual-answer analog stations receive dial tone.

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaClearCall() - Service Request

RetCode_t      cstaClearCall (
    ACSHandle_t   acsHandle,
    InvokeID_t    invokeID,
    ConnectionID_t *call, // deviceID, devIDType are ignored
    PrivateData_t *privateData);

// CSTAClearCallConfEvent - Service Response

typedef struct
{
    ACSHandle_t   acsHandle;
    EventClass_t  eventClass; // CSTACONFIRMATION
    EventType_t   eventType;  // CSTA_CLEAR_CALL_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t  eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
            union
            {
                CSTAClearCallConfEvent_t  clearCall;
            } u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAClearCallConfEvent_t {
    Nulltype      null;
} CSTAClearCallConfEvent_t;

```

Clear Connection Service

Direction: Client to Switch

Function: *cstaClearConnection()*

Confirmation Event: *CSTAClearConnectionConfEvent*

Private Data Function: *attV6ClearConnection()* (private data version 6),
attClearConnection() (private data version 2, 3, 4, and 5)

Service Parameters: *call*

Private Parameters: *dropResource, userInfo*

Ack Parameters: *noData*

Nak Parameter: *universalFailure*

Functional Description:

The Clear Connection Service disconnects the specified device from the designated call. The connection is left in the Null state. The connection identifier is no longer associated with the call. The party to be dropped may be a station or a trunk.

A connection in the alerting state cannot be cleared.

Service Parameters:

call [mandatory] A valid connection identifier that indicates the endpoint to be disconnected.

Private Parameters:

dropResource [optional] Specifies the resource to be dropped from the call. The available resources are DR_CALL_CLASSIFIER and DR_TONE_GENERATOR. The tone generator is any G3 PBX applied denial tone that is timed by the switch.

userInfo [optional] Contains user-to-user information. This parameter allows an application to associate caller information, up to 32 or 96 bytes, with a call. This information may be a customer number, credit card number, alphanumeric digits, or a binary string.

It is propagated with the call when the call is dropped and passed to the application in a Connection Cleared Event Report. A NULL indicates this parameter is not present.

Prior to G3V8, the maximum length of userInfo was 32 bytes. Beginning with G3V8, the maximum length of userInfo was increased to 96 bytes.

⇒ NOTE:

An application using private data version 5 and earlier can only receive a maximum of 32-byte data for userInfo, regardless of the size data that is sent by the switch.

The following UUI protocol types are supported:

- UUI_NONE — There is no data provided in the data parameter.
- UUI_USER_SPECIFIC — The content of the data parameter is a binary string. The correct size (maximum of 32 or 96 bytes) of data must be specified in the size parameter.
- UUI_IA5_ASCII — The content of the data parameter must be a null-terminated IA5 (ASCII) character string. The correct size (maximum of 32 or 96 bytes excluding the null terminator) of data must be specified in the size parameter.

Ack Parameter:

noData None for this service.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- **GENERIC_UNSPECIFIED (0)** The specified data provided for the userInfo parameter exceeds the maximum allowable size. Prior to G3V8, the maximum length of userInfo was 32 bytes. Beginning with G3V8, the maximum length of userInfo was increased to 96 bytes. See the description of the userInfo parameter.
- **INVALID_OBJECT_STATE (22)** The specified connection at the station is not currently active (in alerting or held state) so it cannot be dropped.
- **NO_ACTIVE_CALL (24)** The connectionID contained in the request is invalid. CallID may be incorrect.
- **NO_CONNECTION_TO_CLEAR (27)** The connectionID contained in the request is invalid. CallID may be correct, but deviceID is wrong.
- **RESOURCE_BUSY (33)** The switch is busy with another CSTA request. This can happen when two G3PDs are issuing requests (for example, Hold Call, Retrieve Call, Clear Connection, etc.) to the same device.

Detailed Information:

- **Analog Stations** — The auto-answer analog stations do not receive dial tone after a Clear Connection request. The manual answer analog stations receive dial tone after a Clear Connection request.
- **Bridged Call Appearance** — Clear Connection Service is not permitted on parties in the bridged state and may also be more restrictive if the principal of the bridge has an analog station or the exclusion option is in effect from a station associated with the bridge or PCOL.
- **Drop Button Operation** — The operation of this button is not changed with the Clear Connection Service.
- **Switch Operation** — When a party is dropped from an existing conference call with three or more parties (directly connected to the switch), the other parties remain on the call. Generally, if this was a two-party call, the entire call is dismantled.¹

1. This is the case for typical voice processing. There is a G3 feature "Return VDN Destination" where this is not true. In general, this feature will not be encountered in typical call processing.

Only connected parties can be dropped from a call. Held, bridged, and alerting parties cannot be dropped by the Clear Connection Service.

- Temporary Bridged Appearance — The Clear Connection Service request is denied for a temporary bridged appearance that is not connected on the call.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaClearConnection() - Service Request

RetCode_t      cstaClearConnection (
    ACSHandle_t   acsHandle,
    InvokeID_t    invokeID,
    ConnectionID_t *call, // devIDType= STATIC_ID or
                        // DYNAMIC_ID
    PrivateData_t *privateData);

// CSTAClearConnectionConfEvent - Service Response

typedef struct
{
    ACSHandle_t   acsHandle;
    EventClass_t  eventClass; // CSTACONFIRMATION
    EventType_t   eventType; // CSTA_CLEAR_CONNECTION_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t  eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
            union
            {
                CSTAClearConnectionConfEvent_t  clearConnection;
            } u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAClearConnectionConfEvent_t {
    Nulltype      null;
} CSTAClearConnectionConfEvent_t;
```


Private Data Version 6 Syntax

```

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attV6ClearConnection() - Service Request Private Data
// Setup Function

RetCode_t attV6ClearConnection(
    ATTPrivateData_t *privateData,
    ATTDropResource_t dropResource); // NULL indicates
// no dropResource
// specified
    ATTUserToUserInfo_t *userInfo); // NULL indicates
// no userInfo
// specified

typedef struct ATTPrivateData_t {
    char vendor[32];
    ushort length;
    char data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef enum ATTDropResource_t {
    DR_NONE = -1, // indicates not specified
    DR_CALL_CLASSIFIER = 0, // call classifier to be dropped
    DR_TONE_GENERATOR = 1 // tone generator to be dropped }
ATTDropResource_t;

#define ATT_MAX_USER_INFO 129
#define ATT_MAX_UII_SIZE 96
#define ATTV5_MAX_UII_SIZE 32

typedef struct ATTUserToUserInfo_t {
    ATTUIIProtocolType_t type;
    struct {
        short length; // 0 indicates UII not present

        unsigned char value[ATT_MAX_USER_INFO];
    } data;
} ATTUserToUserInfo_t;

typedef enum ATTUIIProtocolType_t {
    UII_NONE = -1, // indicates not specified
    UII_USER_SPECIFIC = 0, // user-specific
    UII_IA5_ASCII = 4 // null-terminated ascii
// character string
} ATTUIIProtocolType_t;

```

Private Data Version 2-5 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attClearConnection() - Service Request Private Data
// Setup Function

RetCode_t attClearConnection(
    ATTPrivateData_t *privateData,
    ATTDropResource_t dropResource); // NULL indicates
                                     // no dropResource
                                     // specified
    ATTUserToUserInfo_t *userInfo); // NULL indicates
                                     // no userInfo
                                     // specified

typedef struct ATTPrivateData_t {
    char vendor[32];
    ushort length;
    char data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef enum ATTDropResource_t {
    DR_NONE = -1, // indicates not specified
    DR_CALL_CLASSIFIER = 0, // call classifier to be dropped
    DR_TONE_GENERATOR = 1 // tone generator to be dropped
} ATTDropResource_t;

typedef struct ATTUUIProtocolType_t {
    ATTUUIProtocolType_t type;
    struct {
        short length; // 0 indicates UUI not present
        unsigned char value[32];
    } data;
} ATTUUIProtocolType_t;

typedef enum ATTUUIProtocolType_t {
    UUI_NONE = -1, // indicates not specified
    UUI_USER_SPECIFIC = 0, // user-specific
    UUI_IA5_ASCII = 4 // null-terminated ascii
                       // character string
} ATTUUIProtocolType_t;
```

Conference Call Service

Direction: Client to Switch

Function: *cstaConferenceCall ()*

Confirmation Event: *CSTAConferenceCallConfEvent*

Private Data Confirmation Event: *ATTConferenceCallConfEvent* (private data version 5)

Service Parameters: *heldCall, activeCall*

Ack Parameters: *newCall, connList*

Ack Private Parameters: *ucid*

Nak Parameter: *universalFailure*

Functional Description:

This service provides the conference of an existing held call (*heldCall*) and another active or proceeding call (*alerting*, *queued*, *held*, or *connected*) (*activeCall*) at a device provided that *heldCall* and *activeCall* are not both in the *alerting* state at the controlling device. The two calls are merged into a single call and the two connections at the conference-controlling device are resolved into a single connection in the *connected* state. The pre-existing CSTA *connectionID* associated with the device creating the conference is released, and a new *callID* for the resulting conferenced call is provided.

Service Parameters:

heldCall [mandatory] Must be a valid connection identifier for the call that is on hold at the controlling device and is to be conferenced with the activeCall. The deviceID in heldCall must contain the station extension of the controlling device.

activeCall [mandatory] Must be a valid connection identifier for the call that is active or proceeding at the controlling device and that is to be conferenced with the heldCall. The deviceID in activeCall must contain the station extension of the controlling device.

Ack Parameters:

newCall [mandatory — partially supported] A connection identifier specifies the resulting new call identifier for the calls that were conferenced at the conference-controlling device. This connection identifier replaces the two previous call identifiers at that device.

connList [optional — supported] Specifies the devices on the resulting newCall. This includes a count of the number of devices in the new call and a list of up to six connectionIDs and up to six deviceIDs that define each connection in the call.

- If a device is on-PBX, the extension is specified. The extension consists of station or group extensions. Group extensions are provided when the conference is to a group and the conference completes before the call is answered by one of the group members (TEG, PCOL, hunt group, or VDN extension). It may contain alerting extensions.
- The static deviceID of a queued endpoint is set to the split extension of the queue.
- If a party is off-PBX, then its static device identifier or its previously assigned trunk identifier is specified.

Ack Private Parameters:

ucid [optional] Specifies the Universal Call ID (UCID) of newCall. The UCID is a unique call identifier across switches and the network. A valid UCID is a null-terminated ASCII character string. If there is no UCID associated with this call, the ucid contains the ATT_NULL_UCID (a 20-character string of all zeros). This parameter is supported by private data version 5 and later only.

Nak Parameter:

universalFailure

If the request is not successful, the application will receive a `CSTAUniversalFailureConfEvent`. The error parameter in this event may contain the following error values, or one of the error values described in the “`CSTAUniversalFailureConfEvent`” section in Chapter 3:

- `INVALID_CSTA_DEVICE_IDENTIFIER` (12) An invalid device identifier or extension is specified in `heldCall` or `activeCall`.
- `INVALID_CSTA_CONNECTION_IDENTIFIER` (13) The controlling `deviceID` in `heldCall` or `activeCall` has not been specified correctly.
- `GENERIC_STATE_INCOMPATIBILITY` (21) Both calls are alerting or both calls are being service-observed or an active call is in a vector processing stage.
- `INVALID_OBJECT_STATE` (22) The connections specified in the request are not in the valid states for the operation to take place. For example, it does not have one call active and one call in the held state as required.
- `INVALID_CONNECTION_ID_FOR_ACTIVE_CALL` (23) The `callID` or `deviceID` in `activeCall` or `heldCall` has not been specified correctly.
- `RESOURCE_BUSY` (33) The switch is busy with another CSTA request. This can happen when two G3PDs are issuing requests (for example, Hold Call, Retrieve Call, Clear Connection, Conference Call, etc.) to the same device.
- `CONFERENCE_MEMBER_LIMIT_EXCEEDED` (38) The request attempted to add a seventh party to an existing six-party conference call. If a station places a six-party conference call on hold and another party adds yet another station (so that there are again six active parties on the call — the G3 limit), then the station with the call on hold will not be able to retrieve the call.
- `MISTYPED_ARGUMENT_REJECTION` (74) `DYNAMIC_ID` is specified in `heldCall` or `activeCall`.

Detailed Information:

- Analog Stations — Conference Call Service will only be allowed if one call is held and the second is active (talking). Calls on hard-hold or alerting cannot be affected by a Conference Call Service. An analog station will support Conference Call Service even if the “switch-hook flash” field on the G3 system administered form is set to “no”. A “no” in this field disables the switch-hook flash function, meaning that a user cannot conference, hold, or transfer a call from his/her phone set, and cannot have the call waiting feature administered on the phone set.
- Bridged Call Appearance — Conference Call Service is not permitted on parties in the bridged state and may also be more restrictive if the principal of the bridge has an analog station or the exclusion option is in effect from a station associated with the bridge or PCOL.

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaConferenceCall() - Service Request

RetCode_t      cstaConferenceCall (
    ACSHandle_t      acsHandle,
    InvokeID_t      invokeID,
    ConnectionID_t   *heldCall,          // devIDType= STATIC_ID
    ConnectionID_t   *activeCall,       // devIDType= STATIC_ID
    PrivateData_t    *privateData);

// CSTAConferenceCallConfEvent - Service Response

typedef struct
{
    ACSHandle_t      acsHandle;
    EventClass_t     eventClass; // CSTACONFIRMATION
    EventType_t      eventType; // CSTA_CONFERENCE_CALL_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t  eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
            union
            {
                CSTAConferenceCallConfEvent_t  conferenceCall;
            } u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct Connection_t {
    ConnectionID_t party;
    DeviceID_t     staticDevice; // NULL for not present
} Connection_t;

typedef struct ConnectionList_t {
    int             count;
    Connection_t    connection;
} ConnectionList_t;

typedef struct CSTAConferenceCallConfEvent_t {
    ConnectionID_t newCall;
    ConnectionList_t connList;
} CSTAConferenceCallConfEvent_t;

```

Private Data Version 5 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTConferenceCallConfEvent - Service Response Private
// Data (supported by private data version 5 and later only)

typedef struct
{
    ATTEventType_t eventType; // ATT_CONFERENCE_CALL_CONF
    union
    {
        ATTConferenceCallConfEvent_t conferenceCall;
    }u;
} ATTEvent_t;

typedef struct ATTConferenceCallConfEvent_t
{
    ATTUCID_t ucid;
} ATTConferenceCallConfEvent_t;

typedef char ATTUCID_t[64];
```


Consultation Call Service

Direction: Client to Switch

Function: *cstaConsultationCall()*

Confirmation Event: *CSTAConsultationCallConfEvent*

Private Data Function: *attV6ConsultationCall()* (private data version 6),
attConsultationCall() (private data version 2, 3, 4, and 5)

Private Data Confirmation Event: *ATTConsultationCallConfEvent* (private data version 5)

Service Parameters: *activeCall, calledDevice*

Private Parameters: *destRoute, priorityCalling, userInfo*

Ack Parameters: *newCall*

Ack Private Parameters: *ucid*

Nak Parameter: *universalFailure*

Functional Description:

The Consultation Call Service places an existing active call (*activeCall*) at a device on hold and initiates a new call (*newCall*) from the same controlling device. This service provides the compound action of the Hold Call Service followed by Make Call Service. The Consultation Call service has the important special property of associating the G3 Original Call Information from the call being placed on hold with the call being originated. This allows an application running at the consultation desktop to pop a screen using information associated with the call placed on hold. This is an important operation in call centers where an agent calls a specialist for consultation about a call in progress.

The Consultation Call Service request is acknowledged (Ack) by the switch if the switch is able to put the *activeCall* on hold and initiate a new call.

The request is negatively acknowledged if the switch:

- fails to put *activeCall* on hold (for example, call is in alerting state), or
- fails to initiate a new call (for example, invalid parameter).

If the request is negatively acknowledged, the G3PD will attempt to put the *activeCall* to its original state, if the original state is known by the G3PD before the service request. If the original state is unknown, there is no recovery for the *activeCall*'s original state.

Service Parameters:

activeCall [mandatory] A valid connection identifier that indicates the connection to be placed on hold. This party must be in the active (talking) state or already held. The device associated with the activeCall must be a station. If the party specified in the request refers to a trunk device, the request will be denied. The deviceID in activeCall must contain the station extension of the controlling device.

calledDevice [mandatory] Must be a valid on-PBX extension or off-PBX number. On-PBX extension may be a station extension, VDN, split, hunt group, announcement extension, or logical agent's login ID. The calledDevice may include TAC/ARS/AAR information for off-PBX numbers. Trunk Access Code, Authorization Codes, and Force Entry of Account Codes can be specified with the calledDevice as if they were entered from the voice terminal using the keypad.

Private Parameters:

- destRoute*** [optional] Specifies the TAC/ARS/AAR information for an off- PBX destination, if the information is not included in the calledDevice. A NULL indicates this parameter is not specified.
- priorityCalling*** [mandatory] Specifies the priority of the call. Values are On (TRUE) or Off (FALSE). If On is selected, a priority call is attempted for an on-PBX calledDevice. Note that the G3 PBX does not permit priority calls to certain types of extensions (such as VDNs).
- userInfo*** [optional] Contains user-to-user information. This parameter allows an application to associate caller information, up to 32 or 96 bytes, with a call. This information may be a customer number, credit card number, alphanumeric digits, or a binary string.

It is propagated with the call whether the call is made to a destination on the local switch or to a destination on a remote switch over PRI trunks. The switch sends the UUI in the ISDN SETUP message over the PRI trunk to establish the call. The local and the remote switch include the UUI in the Delivered Event Report and in the cstaRouteRequestEvent to the application. A NULL indicates this parameter is not present.

Prior to G3V8, the maximum length of userInfo was 32 bytes. Beginning with G3V8, the maximum length of userInfo was increased to 96 bytes.

⇒ NOTE:

An application using private data version 5 and earlier can only receive a maximum of 32-byte data for userInfo, regardless of the size data that is sent by the switch.

The following UUI protocol types are supported:

- UUI_NONE — There is no data provided in the data parameter.
- UUI_USER_SPECIFIC — The content of the data parameter is a binary string. The correct size (maximum of 32 or 96 bytes) of data must be specified in the size parameter.
- UUI_IA5_ASCII — The content of the data parameter must be a null-terminated IA5 (ASCII) character string. The correct size (maximum of 32 or 96 bytes excluding the null terminator) of data must be specified in the size parameter.

Ack Parameters:

newCall [mandatory] A connection identifier indicates the connection between the controlling device and the new call. The *newCall* parameter contains the callID of the call and the station extension of the controlling device.

Ack Private Parameters:

ucid [optional] Specifies the Universal Call ID (UCID) of *newCall*. The UCID is a unique call identifier across switches and the network. A valid UCID is a null-terminated ASCII character string. If there is no UCID associated with this call, the *ucid* contains the ATT_NULL_UCID (a 20-character string of all zeros). This parameter is supported by private data version 5 and later only.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- GENERIC_UNSPECIFIED (0) The specified data provided for the *userInfo* parameter exceeds the maximum allowable size. Prior to G3V8, the maximum length of *userInfo* was 32 bytes. Beginning with G3V8, the maximum length of *userInfo* was increased to 96 bytes. See the description of the *userInfo* parameter.
- INVALID_CSTA_DEVICE_IDENTIFIER (12) An invalid device identifier or extension is specified in *activeCall*.
- INVALID_CSTA_CONNECTION_IDENTIFIER (13) The connection identifier contained in the request is invalid or does not correspond to a station.
- NO_ACTIVE_CALL (24) The party to be put on hold is not currently active (for example, in alerting state) so it cannot be put on hold.
- GENERIC_STATE_INCOMPATIBILITY (21) (CS0/18) The originator does not go off-hook within five seconds after originating the call and cannot be forced off-hook.

- RESOURCE_BUSY (33) The switch is busy with another CSTA request. This can happen when two G3PDs are issuing requests (for example, Hold Call, Retrieve Call, Clear Connection, etc.) to the same device.
- OUTSTANDING_REQUEST_LIMIT_EXCEEDED (44) The client attempted to put a third party (two parties are on hold already) on hold on an analog station.
- MISTYPED_ARGUMENT_REJECTION (74) DYNAMIC_ID is specified in activeCall.

Detailed Information:

See “Detailed Information:” in the “Hold Call Service” section and “Detailed Information:” in the “Make Predictive Call Service” section in this chapter.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaConsultationCall() - Service Request

RetCode_t      cstaConsultationCall (
    ACSHandle_t   acsHandle,
    InvokeID_t    invokeID,
    ConnectionID_t *activeCall,      // devIDType= STATIC_ID
    DeviceID_t    *calledDevice,
    PrivateData_t *privateData);

// CSTAConsultationCallConfEvent - Service Response

typedef struct
{
    ACSHandle_t   acsHandle;
    EventClass_t  eventClass; // CSTACONFIRMATION
    EventType_t   eventType; // CSTA_CONSULTATION_CALL_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t  eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
            union
            {
                CSTAConsultationCallConfEvent_t consultationCall;
            }
        } u;
    } cstaConfirmation;
} event;
char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAConsultationCallConfEvent_t {
    ConnectionID_t newCall;
} CSTAConsultationCallConfEvent_t;
```

Private Data Version 6 Syntax

```

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attV6ConsultationCall() - Service Request Private Data
// Setup Function

RetCode_t attV6ConsultationCall(
    ATTPrivateData_t*privateData,
    DeviceID_t      *destRoute, // NULL indicates not
specified
    Boolean         priorityCalling; // TRUE = On, FALSE =
Off
    ATTUserToUserInfo_t *userInfo); // NULL indicates not
// specified

typedef struct ATTPrivateData_t {
    char    vendor[32];
    ushort length;
    char    data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

#define ATT_MAX_USER_INFO 129
#define ATT_MAX_UII_SIZE 96
#define ATTV5_MAX_UII_SIZE 32

typedef struct ATTUserToUserInfo_t {
    ATTUIIProtocolType_t type;
    struct {
        short length; // 0 indicates UII not present
        unsigned char value[ATT_MAX_USER_INFO];
    } data;
} ATTUserToUserInfo_t;

typedef enum ATTUIIProtocolType_t {
    UII_NONE          = -1, // indicates not specified
    UII_USER_SPECIFIC = 0, // user-specific
    UII_IA5_ASCII     = 4   // null-terminated ascii
// character string
} ATTUIIProtocolType_t;

```

Private Data Version 6 Syntax (Continued)

```
// ATTConsultationCallConfEvent - Service Response Private
// Data (supported by private data version 5 and later only)

typedef struct
{
    ATTEventType_t eventType;// ATT_CONSULTATION_CALL_CONF
    union
    {
        ATTConsultationCallConfEvent_t consultationCall;
    }u;
} ATTEvent_t;

typedef struct ATTConsultationCallConfEvent_t
{
    ATTUCID_t ucid;
} ATTConsultationCallConfEvent_t;

typedef char ATTUCID_t[64];
```


Private Data Version 2-5 Syntax

```

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attConsultationCall() - Service Request Private Data
// Setup Function

RetCode_t attConsultationCall(
    ATTPrivateData_t *privateData,
    DeviceID_t *destRoute, // NULL indicates not specified
    Boolean priorityCalling; // TRUE = On, FALSE = Off
    ATTUserToUserInfo_t *userInfo); // NULL indicates not
// specified

typedef struct ATTPrivateData_t {
    char vendor[32];
    ushort length;
    char data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef struct ATTV5UserToUserInfo_t {
    ATTUUIProtocolType_t type;
    struct {
        short length; // 0 indicates UUI not present
        unsigned char value[33];
    } data;
} ATTV5UserToUserInfo_t;

typedef enum ATTUUIProtocolType_t {
    UUI_NONE = -1, // indicates not specified
    UUI_USER_SPECIFIC = 0, // user-specific
    UUI_IA5_ASCII = 4 // null-terminated ascii
// character string
} ATTUUIProtocolType_t;

// ATTConsultationCallConfEvent - Service Response Private
// Data (supported by private data version 5 and later only)

typedef struct
{
    ATTEventType_t eventType; // ATT_CONSULTATION_CALL_CONF
    union
    {
        ATTConsultationCallConfEvent_t consultationCall;
    }u;
} ATTEvent_t;

typedef struct ATTConsultationCallConfEvent_t
{
    ATTUCID_t ucid;
} ATTConsultationCallConfEvent_t;

typedef char ATTUCID_t[64];

```

Consultation Direct-Agent Call Service

Direction: Client to Switch

Function: *cstaConsultationCall()*

Confirmation Event: *CSTAConsultationCallConfEvent*

Private Data Function: *attV6DirectAgentCall()* (private data version 6),
attDirectAgentCall() (private data version 2, 3, 4, and 5)

Private Data Confirmation Event: *attConsultationCallConfEvent*

Service Parameters: *activeCall, calledDevice*

Private Parameters: *split, priorityCalling, userInfo*

Ack Parameters: *newCall*

Ack Private Parameters: *ucid*

Nak Parameter: *universalFailure*

Functional Description:

The Consultation Direct-Agent Call Service places an existing active call (*activeCall*) at a device on hold and initiates a new direct-agent call (*newCall*) from the same controlling device. This service provides the compound action of the Hold Call Service followed by Make Direct-Agent Call Service. Like the Consultation Service, the Consultation Direct Agent Call service has the important special property of associating the G3 Original Call Information from the call being placed on hold with the call being originated. This allows an application running at the consultation desktop to pop a screen using information associated with the call placed on hold. This is an important operation in call centers where an agent calls a specialist for consultation about a call in progress.

The Consultation Direct-Agent Call Service request is acknowledged (Ack) by the switch if the switch is able to put the *activeCall* on hold and initiates a new direct-agent call.

The request is negatively acknowledged if the switch:

- Fails to put *activeCall* on hold (for example, call is in alerting state), or
- Fails to initiate a new direct-agent call (for example, invalid parameter).

If the request is negatively acknowledged, the G3PD will attempt to put the *activeCall* into the active state, if it was in the active or held state.

The Consultation Direct Agent Call Service should be used only in the following two situations:

- Consultation Direct Agent Calls in a non-EAS environment
- Consultation Direct Agent Calls in an EAS environment only when it is required to ensure that these calls against a skill other than that skill specified for these measurements on the DEFINITY PBX for that agent.

Prefereably in an EAS environment, Consultation Direct Agent Calls can be made using the Make Call service and specifying an Agent login-ID as the destination device. In this case Consultation Direct Agent Calls will be measured against the skill specified or those measurements on the DEFINITY PBX for that agent.

Service Parameters:

- activeCall*** [mandatory] A valid connection identifier that indicates the connection to be placed on hold. This party must be in the active (talking) state or already held. The device associated with the activeCall must be a station. If the party specified in the request refers to a trunk device, the request will be denied. The deviceID in activeCall must contain the station extension of the controlling device.
- calledDevice*** [mandatory] Must be a valid ACD agent extension. Agent at calledDevice must be logged in.

Private Parameters:

- split*** [mandatory] Contains a valid split extension. Agent at calledDevice must be logged into this split.
- priorityCalling*** [mandatory] Specifies the priority of the call. Values are On (TRUE) or Off (FALSE). If On is selected, a priority call is attempted for an on-PBX calledDevice. Note that the G3 PBX does not permit priority calls to certain types of extensions (such as VDNs).
- userInfo*** [optional] Contains user-to-user information. This parameter allows the application to associate caller information, up to 32 or 96 bytes, with a call. It may be a customer number, credit card number, alphanumeric digits, or a binary string.

It is propagated with the call whether the call is made to a destination on the local switch or to a destination on a remote switch over PRI trunks. The switch sends the UUI in the ISDN SETUP message over the PRI trunk to establish the call. The local and the remote switch include the UUI in the Delivered Event Report and in the cstaRouteRequestEvent to the application. A NULL indicates this parameter is not present.

Prior to G3V8, the maximum length of userInfo was 32 bytes. Beginning with G3V8, the maximum length of userInfo was increased to 96 bytes.

⇒ NOTE:

An application using private data version 5 and earlier can only receive a maximum of 32-byte data for userInfo, regardless of the size data that is sent by the switch.

The following UUI protocol types are supported:

- UUI_NONE — There is no data provided in the data parameter.
- UUI_USER_SPECIFIC — The content of the data parameter is a binary string. The correct size (maximum of 32 or 96 bytes) of data must be specified in the size parameter.
- UUI_IA5_ASCII — The content of the data parameter must be a null-terminated IA5 (ASCII) character string. The correct size (maximum of 32 or 96 bytes excluding the null terminator) of data must be specified in the size parameter.

Ack Parameters:

newCall [mandatory] A connection identifier indicates the connection between the controlling device and the new call. The *newCall* parameter contains the callID of the call and the station extension of the controlling device.

Ack Private Parameters:

ucid [optional] Specifies the Universal Call ID (UCID) of *newCall*. The UCID is a unique call identifier across switches and the network. A valid UCID is a null-terminated ASCII character string. If there is no UCID associated with this call, the *ucid* contains the ATT_NULL_UCID (a 20-character string of all zeros). This parameter is supported by private data version 5 and later only.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- GENERIC_UNSPECIFIED (0) The specified data provided for the *userInfo* parameter exceeds the maximum allowable size. Prior to G3V8, the maximum length of *userInfo* was 32 bytes. Beginning with G3V8, the maximum length of *userInfo* was increased to 96 bytes. See the description of the *userInfo* parameter.
- GENERIC_UNSPECIFIED (0) (CS3/11, CS3/15) Agent is not a member of the split or agent is not currently logged in split.
- VALUE_OUT_OF_RANGE (3) (CS0/100, CS0/96) The split contains an invalid value or invalid information element contents was detected.
- INVALID_CALLING_DEVICE (5) (CS3/27) The *callingDevice* is out of service or not administered correctly in the switch.
- PRIVILEGE_VIOLATION_ON_CALLED_DEVICE (9) (CS0/21, CS0/52) The COR check for completing the call failed. The call was attempted over a trunk that the originator has restricted from use.

- INVALID_CSTA_DEVICE_IDENTIFIER (12) An invalid device identifier or extension is specified in activeCall, the calledDevice is an invalid station extension, or the split does not contain a valid hunt group extension.
- INVALID_CSTA_CONNECTION_IDENTIFIER (13) The connection identifier contained in the request is invalid or does not correspond to a station.
- INVALID_DESTINATION (14) (CS3/24) The call was answered by an answering machine.
- INVALID_OBJECT_TYPE (18) (CS0/58, CS3/80) There is incompatible bearer service for the originating or destination address. For example, the originator is administered as a data hotline station or the destination is a data station. Call options are incompatible with this service.
- GENERIC_STATE_INCOMPATIBILITY (21) (CS0/18) The originator does not go off-hook within five seconds after originating the call and cannot be forced off-hook.
- INVALID_OBJECT_STATE (22) (CS0/98) Request (message) is incompatible with call state
- NO_ACTIVE_CALL (24) The party to be put on hold is not currently active (for example, in alerting state) so it cannot be put on hold.
- RESOURCE_BUSY (33) (CS0/17) The user is busy on another call and cannot originate this call. The switch is busy with another CSTA request. This can happen when two G3PDs are issuing requests (for example, Hold Call, Retrieve Call, Clear Connection, Make Call, etc.) to the same device.
- GENERIC_SUBSCRIBED_RESOURCE_AVAILABILITY (41) (CS0/50) Service or option not subscribed/provisioned (AMD must be enabled).
- OUTSTANDING_REQUEST_LIMIT_EXCEEDED (44) The client attempted to put a third party (two parties are on hold already) on hold on an analog station.
- MISTYPED_ARGUMENT_REJECTION (74) DYNAMIC_ID is specified in activeCall.

Detailed Information:

See “Detailed Information:” in the “Hold Call Service” section and “Detailed Information:” in the “Make Direct-Agent Call Service” section in this chapter.

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaConsultationCall() - Service Request

RetCode_t      cstaConsultationCall (
    ACSHandle_t   acsHandle,
    InvokeID_t    invokeID,
    ConnectionID_t *activeCall,    // devIDType= STATIC_ID
    DeviceID_t    *calledDevice,
    PrivateData_t *privateData);

// CSTAConsultationCallConfEvent - Service Response

typedef struct
{
    EventType_t    eventType; // CSTA_CONSULTATION_CALL_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t    eventHeader;
    union
    {
        struct
        {
            InvokeID_t    invokeID;
            union
            {
                CSTAConsultationCallConfEvent_t consultationCall;
            }
        } u;
    } cstaConfirmation;
} event;
char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAConsultationCallConfEvent_t {
    ConnectionID_t newCall;
} CSTAConsultationCallConfEvent_t;

```

Private Data Version 6 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attV6DirectAgentCall() - Service Request Private Data
// Setup Function

RetCode_t attV6DirectAgentCall(
    ATTPrivateData_t*privateData,
    DeviceID_t *split, // NULL indicates not
specified
    Boolean priorityCalling;// TRUE = On, FALSE =
Off
    ATTUserToUserInfo_t *userInfo); // NULL indicates not
// specified

typedef struct ATTPrivateData_t {
    char vendor[32];
    ushort length;
    char data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

#define ATT_MAX_USER_INFO 129
#define ATT_MAX_UII_SIZE 96
#define ATTV5_MAX_UII_SIZE 32

typedef struct ATTUserToUserInfo_t {
    ATTUIIProtocolType_t type;
    struct {
        short length; // 0 indicates UII not present
        unsigned char value[ATT_MAX_USER_INFO];
    } data;
} ATTUserToUserInfo_t;

typedef enum ATTUIIProtocolType_t {
    UII_NONE = -1, // indicates not specified
    UII_USER_SPECIFIC = 0, // user-specific
    UII_IA5_ASCII = 4 // null-terminated ascii
// character string
} ATTUIIProtocolType_t;
```


Private Data Version 6 Syntax (Continued)

```
// ATTConsultationCallConfEvent - Service Response Private
// Data (supported by private data version 5 and later only)

typedef struct
{
    ATTEventType_t  eventType;// ATT_CONSULTATION_CALL_CONF
    union
    {
        {
            ATTConsultationCallConfEvent_t  consultationCall;
        }u;
    } ATTEvent_t;

typedef struct ATTConsultationCallConfEvent_t
{
    ATTUCID_t  ucid;
} ATTConsultationCallConfEvent_t;

typedef char ATTUCID_t[64];
```

Private Data Version 2-5 Syntax

```

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attDirectAgentCall() - Service Request Private Data
// Setup Function

RetCode_t attDirectAgentCall(
    ATTPrivateData_t *privateData,
    DeviceID_t *split, // NULL indicates not specified
    Boolean priorityCalling; // TRUE = On, FALSE = Off
    ATTUserToUserInfo_t *userInfo); // NULL indicates not
// specified

typedef struct ATTPrivateData_t {
    char vendor[32];
    ushort length;
    char data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef struct ATTV5UserToUserInfo_t {
    ATTUUIProtocolType_t type;
    struct {
        short length; // 0 indicates UUI not present
        unsigned char value[33];
    } data;
} ATTV5UserToUserInfo_t;

typedef enum ATTUUIProtocolType_t {
    UUI_NONE = -1, // indicates not specified
    UUI_USER_SPECIFIC = 0, // user-specific
    UUI_IA5_ASCII = 4 // null-terminated ascii
// character string
} ATTUUIProtocolType_t;

// ATTConsultationCallConfEvent - Service Response Private
// Data (supported by private data version 5 and later only)

typedef struct
{
    ATTEventType_t eventType; // ATT_CONSULTATION_CALL_CONF
    union
    {
        ATTConsultationCallConfEvent_t consultationCall;
    }u;
} ATTEvent_t;

typedef struct ATTConsultationCallConfEvent_t
{
    ATTUCID_t ucid;
} ATTConsultationCallConfEvent_t;

typedef char ATTUCID_t[64];

```

Consultation Supervisor-Assist Call Service

Direction: Client to Switch

Function: *cstaConsultationCall()*

Confirmation Event: *CSTAConsultationCallConfEvent*

Private Data Function: *attV6SupervisorAssistCall()* (private data version 6),
attSupervisorAssistCall() (private data version 2, 3, 4, and 5)

Private Data Confirmation Event: *attConsultationCallConfEvent*

Service Parameters: *activeCall, calledDevice*

Private Parameters: *split, userInfo*

Ack Parameters: *newCall*

Ack Private Parameters: *ucid*

Nak Parameter: *universalFailure*

Functional Description:

The Consultation Supervisor-Assist Call Service places an existing active call (*activeCall*) at a device on hold and initiates a new supervisor-assist call (*newCall*) from the same controlling device. This service provides the compound action of the Hold Call Service followed by Make Supervisor-Assist. Like the Consultation Service, the Consultation Supervisor-Assist Call service has the important special property of associating the G3 Original Call Information from the call being placed on hold with the call being originated. This allows an application running at the consultation desktop to pop a screen using information associated with the call placed on hold. This is an important operation in call centers where an agent calls a specialist for consultation about a call in progress.

The Consultation Supervisor-Assist Call Service request is acknowledged (Ack) by the switch if the switch is able to put the *activeCall* on hold and initiates a new supervisor-assist call.

The request is negatively acknowledged if the switch:

- Fails to put *activeCall* on hold (for example, call is in alerting state), or
- Fails to initiate a new direct-agent call (for example, invalid parameter).

If the request is negatively acknowledged, the G3PD will attempt to put the *activeCall* into the active state, if it was in the active or held state.

Service Parameters:

activeCall [mandatory] A valid connection identifier that indicates the connection to be placed on hold. This party must be in the active (talking) state or already held. The device associated with the activeCall must be a station. If the party specified in the request refers to a trunk device, the request will be denied. The deviceID in activeCall must contain the station extension of the controlling device.

calledDevice [mandatory] Must be a valid ACD agent extension. Agent at calledDevice must be logged in.

Private Parameters:

- split*** [mandatory] Contains a valid split extension. Agent at calledDevice must be logged into this split.
- userInfo*** [optional] Contains user-to-user information. This parameter allows the application to associate caller information, up to 32 or 96 bytes, with a call. It may be a customer number, credit card number, alphanumeric digits, or a binary string.

It is propagated with the call whether the call is made to a destination on the local switch or to a destination on a remote switch over PRI trunks. The switch sends the UUI in the ISDN SETUP message over the PRI trunk to establish the call. The local and the remote switch include the UUI in the Delivered Event Report and in the cstaRouteRequestEvent to the application. A NULL indicates this parameter is not present.

Prior to G3V8, the maximum length of userInfo was 32 bytes. Beginning with G3V8, the maximum length of userInfo was increased to 96 bytes.

NOTE:

An application using private data version 5 and earlier can only receive a maximum of 32-byte data for userInfo, regardless of the size data that is sent by the switch.

The following UUI protocol types are supported:

- UUI_NONE — There is no data provided in the data parameter.
- UUI_USER_SPECIFIC — The content of the data parameter is a binary string. The correct size (maximum of 32 or 96 bytes) of data must be specified in the size parameter.
- UUI_IA5_ASCII — The content of the data parameter must be a null-terminated IA5 (ASCII) character string. The correct size (maximum of 32 or 96 bytes excluding the null terminator) of data must be specified in the size parameter.

Ack Parameters:

- newCall*** [mandatory] A connection identifier indicates the connection between the controlling device and the new call. The newCall parameter contains the callID of the call and the station extension of the controlling device.

Ack Private Parameters:

ucid [optional] Specifies the Universal Call ID (UCID) of newCall. The UCID is a unique call identifier across switches and the network. A valid UCID is a null-terminated ASCII character string. If there is no UCID associated with this call, the *ucid* contains the ATT_NULL_UCID (a 20-character string of all zeros). This parameter is supported by private data version 5 and later only.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- **GENERIC_UNSPECIFIED (0)** The specified data provided for the *userInfo* parameter exceeds the maximum allowable size. Prior to G3V8, the maximum length of *userInfo* was 32 bytes. Beginning with G3V8, the maximum length of *userInfo* was increased to 96 bytes. See the description of the *userInfo* parameter.
- **GENERIC_UNSPECIFIED (0) (CS3/11, CS3/15)** The agent is not a member of the split or the agent is not currently logged in split.
- **VALUE_OUT_OF_RANGE (3) (CS0/100, CS0/96)** The split contains an invalid value or invalid information element contents was detected.
- **INVALID_CALLING_DEVICE (5) (CS3/27)** The *callingDevice* is out of service or not administered correctly in the switch.
- **PRIVILEGE_VIOLATION_ON_CALLED_DEVICE (9) (CS0/21, CS0/52)** The COR check for completing the call failed. The call was attempted over a trunk that the originator has restricted from use.
- **INVALID_CSTA_DEVICE_IDENTIFIER (12)** An invalid device identifier or extension is specified in *activeCall*, the *calledDevice* is an invalid station extension, or the split does not contain a valid hunt group extension.
- **INVALID_CSTA_CONNECTION_IDENTIFIER (13)** The connection identifier contained in the request is invalid or does not correspond to a station.
- **INVALID_DESTINATION (14) (CS3/24)** The call was answered by an answering machine.

- INVALID_OBJECT_TYPE (18) (CS0/58, CS3/80) There is incompatible bearer service for the originating or destination address. For example, the originator is administered as a data hotline station or the destination is a data station. Call options are incompatible with this service.
- GENERIC_STATE_INCOMPATIBILITY (21) (CS0/18) The originator does not go off-hook within five seconds after originating the call and cannot be forced off-hook.
- INVALID_OBJECT_STATE (22) (CS0/98) Request (message) is incompatible with the call state.
- NO_ACTIVE_CALL (24) The party to be put on hold is not currently active (for example, in alerting state) so it cannot be put on hold.
- RESOURCE_BUSY (33) (CS0/17) The user is busy on another call and cannot originate this call. The switch is busy with another CSTA request. This can happen when two G3PDs are issuing requests (for example, Hold Call, Retrieve Call, Clear Connection, Make Call, etc.) to the same device.
- GENERIC_SUBSCRIBED_RESOURCE_AVAILABILITY (41) (CS0/50) Service or option not subscribed/provisioned (AMD must be enabled).
- OUTSTANDING_REQUEST_LIMIT_EXCEEDED (44) The client attempted to put a third party on hold on an analog station when two parties are already on hold.
- MISTYPED_ARGUMENT_REJECTION (74) DYNAMIC_ID is specified in activeCall.

Detailed Information:

See “Detailed Information:” in the “Hold Call Service” section and “Detailed Information:” in the “Make Direct-Agent Call Service” section in this chapter.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaConsultationCall() - Service Request

RetCode_t      cstaConsultationCall (
    ACSHandle_t    acsHandle,
    InvokeID_t     invokeID,
    ConnectionID_t *activeCall,      // devIDType= STATIC_ID
    DeviceID_t     *calledDevice,
    PrivateData_t  *privateData);

// CSTAConsultationCallConfEvent - Service Response

typedef struct
{
    ACSHandle_t    acsHandle;
    EventClass_t   eventClass; // CSTACONFIRMATION
    EventType_t    eventType; // CSTA_CONSULTATION_CALL_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t  eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
            union
            {
                CSTAConsultationCallConfEvent_t consultationCall;
            }
        } u;
    } cstaConfirmation;
} event;
char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAConsultationCallConfEvent_t {
    ConnectionID_t newCall;
} CSTAConsultationCallConfEvent_t;
```


Private Data Version 6 Syntax

```

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attV6SupervisorAssistCall() - Service Request Private
Data
                                // Setup Function

RetCode_t    attV6SupervisorAssistCall(
    ATTPrivateData_t*privateData,
    DeviceID_t    *split,        // mandatory
                                // NULL indicates not
specified
    ATTUserToUserInfo_t *userInfo); // NULL indicates not
                                // specified

typedef struct ATTPrivateData_t {
    char    vendor[32];
    ushort length;
    char    data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

#define ATT_MAX_USER_INFO 129
#define ATT_MAX_UII_SIZE 96
#define ATTV5_MAX_UII_SIZE 32

typedef struct ATTUserToUserInfo_t {
    ATTUIIProtocolType_t type;
    struct {
                                short length; // 0 indicates UII not present
                                unsigned char value[ATT_MAX_USER_INFO];
    } data;
} ATTUserToUserInfo_t;

typedef enum ATTUIIProtocolType_t {
    UII_NONE            = -1, // indicates not specified
    UII_USER_SPECIFIC  = 0, // user-specific
    UII_IA5_ASCII      = 4   // null-terminated ascii
                                // character string
} ATTUIIProtocolType_t;

```

Private Data Version 6 Syntax (Continued)

```
// ATTConsultationCallConfEvent - Service Response Private
// Data (supported by private data version 5 and later only)

typedef struct
{
    ATTEventType_t eventType;// ATT_CONSULTATION_CALL_CONF
    union
    {
        ATTConsultationCallConfEvent_t consultationCall;
    }u;
} ATTEvent_t;

typedef struct ATTConsultationCallConfEvent_t
{
    ATTUCID_t ucid;
} ATTConsultationCallConfEvent_t;

typedef char ATTUCID_t[64];
```

Private Data Version 2-5 Syntax

```

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attSupervisorAssistCall() - Service Request Private Data
// Setup Function

RetCode_t attSupervisorAssistCall(
    ATTPrivateData_t *privateData,
    DeviceID_t *split, // mandatory
                        // NULL indicates not specified
    ATTUserToUserInfo_t *userInfo); // NULL indicates not
                                    // specified

typedef struct ATTPrivateData_t {
    char vendor[32];
    ushort length;
    char data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef struct ATTV5UserToUserInfo_t {
    ATTUUIProtocolType_t type;
    struct {
        short length; // 0 indicates UUI not present
        unsigned char value[33];
    } data;
} ATTV5UserToUserInfo_t;

typedef enum ATTUUIProtocolType_t {
    UUI_NONE = -1, // indicates not specified
    UUI_USER_SPECIFIC = 0, // user-specific
    UUI_IA5_ASCII = 4 // null-terminated ascii
                      // character string
} ATTUUIProtocolType_t;

// ATTConsultationCallConfEvent - Service Response Private
// Data (supported by private data version 5 and later only)

typedef struct
{
    ATTEventType_t eventType; // ATT_CONSULTATION_CALL_CONF
    union
    {
        {
            ATTConsultationCallConfEvent_t consultationCall;
        }u;
    }
} ATTEvent_t;

typedef struct ATTConsultationCallConfEvent_t
{
    ATTUCID_t ucid;
} ATTConsultationCallConfEvent_t;

typedef char ATTUCID_t[64];

```

Deflect Call Service

Direction: Client to Switch
Function: *cstaDeflectCall ()*
Confirmation Event: *CSTADeflectCallConfEvent*
Service Parameters: *deflectCall, calledDevice*
Ack Parameters: *noData*
Nak Parameter: *universalFailure*

Functional Description:

This service redirects an alerting call at a device to a new destination, either on-PBX or off-PBX. The call at the redirecting device is dropped after a successful redirection. An application may redirect an alerting call (at different devices) any number of times until the call is answered or dropped by the caller.

The service request is positively acknowledged if the call has successfully redirected for an on- PBX destination. For an off-PBX destination, this does not imply a successful redirection. It indicates that the switch attempted to redirect the call to the off-PBX destination and subsequent call progress events or tones may indicate redirection success or failure.

If the service request is negatively acknowledged, the call remains at the redirecting device and the calledDevice is not involved in the call.

Service Parameters:

deflectCall [mandatory] Specifies the connectionID of the call that is to be redirected to another destination. The call must be in the alerting state at the device. The device must be a valid voice station extension.

calledDevice [mandatory] Specifies the destination to which the call is redirected. The destination can be an on-PBX or off-PBX endpoint. For on-PBX endpoints, the calledDevice may be stations, queues, announcements, VDNs, or logical agent extensions.

Ack Parameter:

noData None for this service.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- PRIVILEGE_VIOLATION_ON_CALLED_DEVICE (9) (CS3/42)
 - Attempted to redirect a call back to the call originator or to the redirecting device itself.
 - Attempted to redirect a call on the calledDevice of a cstaMakePredictiveCall.
- INVALID_OBJECT_STATE (22) (3/63)
 - An invalid callID or device identifier is specified in deflectCall.
 - The deflectCall is not in alerting state.
 - Attempted to redirect the call while in vector processing.

- PRIVILEGE_VIOLATION_ON_SPECIFIED_DEVICE (8) (CS3/43)

The request may fail because of one of the following:

- invalid destination specified
 - toll restrictions on destination
 - COR restrictions on destination
 - destination is remote access extension
 - call origination restriction on the redirecting device
 - call is in vector processing
- RESOURCE_BUSY (33) (CS0/17) A call redirected to a busy station, a station that has call forwarding active, or a TEG group with one or more members busy will be rejected with this error.
 - GENERIC_SUBSCRIBED_RESOURCE_AVAILABILITY (41) (CS0/50) This service is requested on a G3 PBX administered as a release earlier than G3V4.
 - GENERIC_OPERATION (1) (CS0/111) This service is requested on a queued call or protocol error in the request.

Detailed Information:

- Administration Without Hardware — A call cannot be redirected to/from an AWOH station. However, if the AWOH station is forwarded to a real physical station, the call can be redirected to/from such a station, if it is being alerted.
- Attendants — Calls on attendants cannot be redirected.
- Auto Call Back — ACB calls cannot be redirected by the cstaDeflectCall service from the call originator.
- Bridged Call Appearance — A call may be redirected away from a primary extension or from a bridged station. When that happens, the call is redirected away from the primary and all bridged stations.
- Call Waiting — A call may be redirected while waiting at a busy analog set.
- Deflect From Queue — This service will not redirect a call from a queue to a new destination.
- Delivered Event — If the calling device or call is monitored, an application subsequently receives Delivered (or Network Reached) Event when redirection succeeds.

- **Diverted Event** — If the redirecting device is monitored by a `cstaMonitorDevice` or the call is monitored by a `cstaMonitorCallsViaDevice`, it will receive a Diverted Event when the call is successfully redirected, but there will be no Diverted Event for a `cstaMonitorCall` association.
- **Loop Back** — A call cannot be redirected to the call originator or to the redirecting device itself.
- **Off-PBX Destination** — If the call is redirected to an off-PBX destination, the caller will hear call progress tones. There may be conditions (for example, trunk not available) that will prevent the call from being placed. The call is nevertheless routed in those cases, and the caller receives busy or reorder treatment. An application may subsequently receive Failed, Call Cleared, or Connection Cleared Events if redirection fails.

If trunk-to-trunk transfer is disallowed by the switch administration, redirection of an incoming trunk call to an off-PBX destination will fail.

- **Priority and Forwarded Calls** — Priority and forwarded calls are allowed to be redirected with `cstaDeflectCall`.
- **Service Availability**— This service is only available on a G3 PBX with G3V4 or later software.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaDeflectCall() - Service Request

RetCode_t      cstaDeflectCall (
    ACSHandle_t      acsHandle,
    InvokeID_t       invokeID,
    ConnectionID_t   *deflectCall,
    DeviceID_t       *calledDevice,
    PrivateData_t    *privateData);

// CSTADeflectCallConfEvent - Service Response

typedef struct
{
    ACSHandle_t      acsHandle;
    EventClass_t     eventClass; // CSTACONFIRMATION
    EventType_t      eventType; // CSTA_DEFLECT_CALL_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t  eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
            union
            {
                CSTADeflectCallConfEvent_t      deflectCall;
            } u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTADeflectCallConfEvent_t {
    Nulltype      null;
} CSTADeflectCallConfEvent_t;
```


Hold Call Service

Direction: Client to Switch
Function: *cstaHoldCall ()*
Confirmation Event: *CSTAHoldCallConfEvent*
Service Parameters: *activeCall, reservation*
Ack Parameters: *noData*
Nak Parameter: *universalFailure*

Functional Description:

The Hold Call Service places a call on hold at a PBX station. The effect is as if the specified party depressed the hold button on his or her multifunction station to locally place the call on hold or switch-hook flashed on an analog station.

Service Parameters:

activeCall [mandatory] A valid connection identifier that indicates the connection to be placed on hold. This party must be in the active (talking) state or already held. The device associated with the activeCall must be a station. If the party specified in the request refers to a trunk device, the request will be denied. The deviceID in activeCall must contain the station extension of the controlling device.

reservation [optional — not supported] Specifies whether the facility is reserved for reuse by the held call. The G3 switch always allows a party to reconnect to a held call. It is recommended that the application always supply TRUE. In actuality, the G3PD ignored the application-supplied value for this parameter.

Ack Parameter:

noData None for this service.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- INVALID_CSTA_DEVICE_IDENTIFIER (12) An invalid device identifier or extension is specified in activeCall.
- INVALID_CSTA_CONNECTION_IDENTIFIER (13) The connection identifier contained in the request is invalid or does not correspond to a station.
- NO_ACTIVE_CALL (24) The party to be put on hold is not currently active (for example, in alerting state) so it cannot be put on hold.
- RESOURCE_BUSY (33) The switch is busy with another CSTA request. This can happen when two G3PDs are issuing requests (for example, Hold Call, Retrieve Call, Clear Connection, etc.) to the same device.
- OUTSTANDING_REQUEST_LIMIT_EXCEEDED (44) The client attempted to put a third party on hold (two parties are on hold already) on an analog station.
- MISTYPED_ARGUMENT_REJECTION (74) DYNAMIC_ID is specified in activeCall.

Detailed Information:

- **Analog Stations** — An analog station cannot switch between a soft-held call and an active call from the voice set. However, with Hold Call Service, this is possible by placing the active call on hard-hold and retrieving the soft-held call. Hold Call Service places a call on conference and/or transfer hold. If that device already had a conference and/or transfer held call and a Hold Call Service is requested, the active call will be placed on hard-hold (unless there is call-waiting, in which case the request is denied).

A maximum of two calls may be in a held state at the same time. A request to have a third call on hold on the same analog station will be denied.

- **Bridged Call Appearance** — Hold Call Service is not permitted on parties in the bridged state and may also be more restrictive if the principal of the bridge has an analog station or the exclusion option is in effect from a station associated with the bridge or PCOL.
- **Busy Verification of Terminals** — A Hold Call Service request will be denied if requested for the verifying user's station.
- **Held State** — If the party is already on hold on the specified call when the switch receives the request, a positive request acknowledgment is returned.
- **Music on Hold** — Music on Hold (if administered and available) will be given to a party placed on hold from the other end either manually or via the Hold Call Service.
- **Switch Operation** — After a party is placed on hold through a Hold Call Service request, the user will not receive dial tone regardless of the type of phone device. Thus, subsequent calls must be placed by selecting an idle call appearance or through the Make Call Service request.

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaHoldCall() - Service Request

RetCode_t      cstaHoldCall (
    ACSHandle_t      acsHandle,
    InvokeID_t      invokeID,
    ConnectionID_t  *activeCall, // devIDType = STATIC_ID
    Boolean          reservation, // not supported - defaults to On
    DeviceID_t      *calledDevice,
    PrivateData_t   *privateData);

// CSTAHoldCallConfEvent - Service Response

typedef struct
{
    ACSHandle_t      acsHandle;
    EventClass_t     eventClass; // CSTACONFIRMATION
    EventType_t      eventType; // CSTA_HOLD_CALL_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t  eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
            union
            {
                CSTAHoldCallConfEvent_t      holdCall;
            } u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAHoldCallConfEvent_t {
    Nulltype      null;
} CSTAHoldCallConfEvent_t;

```

Make Call Service

Direction: Client to Switch

Function: *cstaMakeCall()*

Confirmation Event: *CSTAMakeCallConfEvent*

Private Data Function: *attV6MakeCall()* (private data version 6),
attMakeCall() (private data version 2, 3, 4, and 5)

Private Data Confirmation Event: *ATTMakeCallConfEvent*

Service Parameters: *callingDevice, calledDevice*

Private Parameters: *destRoute, priorityCalling, userInfo*

Ack Parameters: *newCall*

Ack Private Parameters: *ucid*

Nak Parameter: *universalFailure*

Functional Description:

The Make Call Service originates a call between two devices. The service attempts to create a new call and establish a connection with the originating device (*callingDevice*). The Make Call Service also provides a connection identifier (*newCall*) that indicates the connection of the originating device in the *CSTAMakeCallConfEvent*.

The client application uses this service to set up a call on behalf of a station extension (*calling party*) to either an on- or off-PBX endpoint (*calledDevice*). This service can be used by many types of applications such as Office Automation, Messaging, and Outbound Call Management (OCM) for Preview Dialing.

All trunk types (including ISDN-PRI) are supported as facilities for reaching called endpoints for outbound *cstaMakeCall* calls. Call progress feedback is reported as events to the application via Monitor Services. Answer Supervision or Call Classifier is not used for this service.

For the originator to place the call, the *callingDevice* (display or voice) must have an available call appearance for call origination and must not be in the talking (active) state on any call appearances. The originator is allowed to have a call(s) on hold or alerting at the device.

For a digital voice terminal without a speakerphone, when the switch selects the available call appearance for call origination, the red and green status lamps of the call appearance will light. The originator must go off-hook within five seconds. If the call is placed for an analog station without a speakerphone (or a handset), the user must either be idle or off-hook with dial tone, or go off-hook within five seconds after the Make Call request. In either case, the request will be denied if the station fails to go off-hook within five seconds.

The originator may go off-hook and receive dial tone first, and then issue the Make Call Service request for that station. The switch will originate the call on the same call appearance and callID to establish the call.

If the originator is off-hook busy, the call cannot be placed and the request is denied (RESOURCE_BUSY). If the originator is unable to originate for other reasons (see the Nak parameter universalFailure), the switch denies the request.

Service Parameters:

<i>callingDevice</i>	[mandatory] Must be a valid station extension or an AWOH station extension (for phantom calls). ¹
<i>calledDevice</i>	[mandatory] Must be a valid on-PBX extension or off-PBX number. On-PBX extension may be a station extension, VDN, split, hunt group, announcement extension, or logical agent's login ID. The calledDevice may include TAC/ARS/AAR information for off-PBX numbers. Trunk Access Code, Authorization Codes, and Force Entry of Account Codes can be specified with the calledDevice as if they were entered from the voice terminal using the keypad.

-
1. For DEFINITY ECS switch software Release 6.3 and later, a call can be originated from an AWOH station or some group extensions (i.e., a plain [non-ACD] hunt group). This is termed a *phantom call*. Most calls that can be requested for a physical extension can also be requested for an AWOH station and the associated event will also be received. If the call is made on behalf of a group extension, this may not apply. For a detailed description of the phantom call switch feature, refer to CallVisor technical Reference (555-230-220).

Private Parameters:

- destRoute** [optional] Specifies the TAC/ARS/AAR information for an off-PBX destination, if the information is not included in the calledDevice. A NULL indicates that this parameter is not specified.
- priorityCalling** [mandatory] Specifies the priority of the call. Values are "On" (TRUE) or "Off" (FALSE). If On is selected, a priority call is attempted for an on-PBX calledDevice. Note that the G3 PBX does not permit priority calls to certain types of extensions (such as VDNs).
- userInfo** [optional] Contains user-to-user information. This parameter allows the application to associate caller information, up to 32 or 96 bytes, with a call. It may be a customer number, credit card number, alphanumeric digits, or a binary string.

It is propagated with the call whether the call is made to a destination on the local switch or to a destination on a remote switch over PRI trunks. The switch sends the UUI in the ISDN SETUP message over the PRI trunk to establish the call. The local and the remote switch include the UUI in the Delivered Event Report and in the cstaRouteRequestEvent to the application. A NULL indicates this parameter is not present.

Prior to G3V8, the maximum length of userInfo was 32 bytes. Beginning with G3V8, the maximum length of userInfo was increased to 96 bytes.

⇒ NOTE:

An application using private data version 5 and earlier can only receive a maximum of 32-byte data for userInfo, regardless of the size data that is sent by the switch.

The following UUI protocol types are supported:

- UUI_NONE — There is no data provided in the data parameter.
- UUI_USER_SPECIFIC — The content of the data parameter is a binary string. The correct size (maximum of 32 or 96 bytes) of data must be specified in the size parameter.
- UUI_IA5_ASCII — The content of the data parameter must be a null-terminated IA5 (ASCII) character string. The correct size (maximum of 32 or 96 bytes excluding the null terminator) of data must be specified in the size parameter.

Issue 1 — December 2001

Ack Parameters:

newCall [mandatory] A connection identifier that indicates the connection between the originating device and the call. The *newCall* parameter contains the callID of the call and the station extension of the *callingDevice*.

Ack Private Parameters:

ucid [optional] Specifies the Universal Call ID (UCID) of *newCall*. The UCID is a unique call identifier across switches and the network. A valid UCID is a null-terminated ASCII character string. If there is no UCID associated with this call, the *ucid* contains the ATT_NULL_UCID (a 20-character string of all zeros). This parameter is supported by private data version 5 and later only.

Nak Parameter:

A Make Call request will be denied if the request fails before the call is attempted by the PBX.

universalFailure If the request is not successful, the application will receive a *CSTAUniversalFailureConfEvent*. The error parameter in this event may contain the following error values, or one of the error values described in the “*CSTAUniversalFailureConfEvent*” section in Chapter 3:

- **GENERIC_UNSPECIFIED (0)** The specified data provided for the *userInfo* parameter exceeds the maximum allowable size. Prior to G3V8, the maximum length of *userInfo* was 32 bytes. Beginning with G3V8, the maximum length of *userInfo* was increased to 96 bytes. See the description of the *userInfo* parameter.
- **INVALID_CALLING_DEVICE (5)** The *callingDevice* is out of service or not administered correctly in the switch.
- **INVALID_CSTA_DEVICE_IDENTIFIER (12)** An invalid device identifier or extension is specified in *callingDevice*.
- **GENERIC_STATE_INCOMPATIBILITY (21)** The originator does not go off-hook within five seconds after originating the call and cannot be forced off-hook.
- **RESOURCE_BUSY (33)** The user is busy on another call and cannot originate this call, or the switch is busy with another CSTA request. This can happen when two G3PDs are issuing requests (for example, Hold Call, Retrieve Call, Clear Connection, Make Call, etc.) to the same device.

Detailed Information:

- VDN — Priority calls cannot be made to VDNs. Do not set priorityCalling to TRUE when the calledDevice is a VDN.
- AAR/ARS — The AAR/ARS features are accessible by an application through Make Call Service. The calledDevice may include TAC/ARS/AAR information for off-PBX numbers (the switch uses only the first 32 digits as the number). However, it is recommended that, in situations where multiple applications (TSAPI applications and other applications) use ARS trunks, ARS Routing Plans be administered using partitioning to guarantee use of certain trunks to the Telephony Services API application. Each partition should be dedicated to a particular application (this is enforced by the switch).

If the application wants to obtain trunk availability information when ARS/AAR is used (in the calledDevice), it must query the switch about all trunk groups in the ARS partition dedicated. The application may not use the ARS/AAR code in the query to obtain trunk availability information.

When using ARS/AAR, the switch does not tell the application which particular trunk group was selected for a given call.

Care must be given to the proper administration of this feature, particularly the FRLs. If these are not properly assigned, calls may be denied despite trunk availability.

The switch does not attempt to validate the ARS/AAR code prior to placing the call.

ARS must be subscribed in the G3 switch if outbound calls are made over ISDN-PRI facilities.

- ACD Destination — When the destination is an agent login ID or an ACD split, ACD call delivery rules apply. If an ACD agent's extension is specified in the calledDevice, the call is delivered to that ACD agent as a personal call, not a direct agent call.
- ACD Originator — Make Call Service cannot have an ACD Split as the callingDevice.
- Analog Stations — A maximum of three calls (one soft-held, one hard-held, and one active²) may be present at the same time at an analog station. In addition, the station may have a call waiting call.

A request to have more than three calls present will be denied. For example, if an analog station user has three calls present and another call waiting, the user cannot place the active call on hold or answer the call. The only operations allowed are drop the active call or transfer/conference the soft-held and active waiting call.

2. An active party/connection/call is a party/connection/call at the connected state. The user of an active party/connection/call usually has an active talk path and is talking or listening on the call.

- **Announcement Destination** — Announcement calledDevices are treated like on-PBX station users.
- **Attendants** — The attendant group is not supported with Make Call Service. It may never be specified as the callingDevice and in some cases cannot be the calledDevice.
- **Authorization Codes** — If applicable, the originator will be prompted for authorization codes through the phone. The access codes and authorization codes can also be included in the calledDevice, if applicable, as if they were entered from the originator's voice terminal.
- **Bridged Call Appearance** — Make Call Service will always originate the call at the primary extension number of a user having a bridged appearance. For a call to originate at the bridged call appearance of a primary extension, that user must be off-hook at that bridged appearance at the time the Make Call Service is requested.
- **Call Classification** — All call-progress audible tones are provided to the originating user at the calling device (except that the user does not hear dial tone or touch tones). For OCM preview dialing applications, final call classification is done by the station user staffing the callingDevice (who hears call progress tones and manually records the result). If the call was placed to a VDN extension, the originator will hear whatever has been programmed for the vector associated with that VDN.
- **Call Coverage Path Containing VDNs** — Make Call Service is permitted to follow the VDN in the coverage path, provided that the coverage criteria has been met.
- **Call Destination** — If the calledDevice is on-PBX station, the user at the station will receive alerting. The user is alerted according to the call type (ACD or normal). Call delivery depends on the call type, station type, station administered options (manual/auto answer, call waiting, etc.), and station's talk state.

For example, for an ACD call, if the user is off-hook idle, and in auto-answer mode, the call is cut-through immediately. If the user is off-hook busy and has a multifunction- function set, the call will alert a free appearance. If the user is off-hook busy and has an analog set, and the user has "call waiting", the analog station user is given the "call waiting tone". If the user is off-hook busy on an analog station and does not have "call waiting", the calling endpoint will hear busy. If the user is off-hook, alerting is started.

- **Call Forwarding All Calls** — A Make Call Service to a station (calledDevice) with the Call Forwarding All Calls feature active will redirect to the "forwarded to" station.
- **Class of Restrictions (COR)** — The Make Call Service is originated by using the originator's COR. A call placed to a called endpoint whose COR does not allow the call to end will return intercept treatment to the calling endpoint and the Failed Event Report with the error PRIVILEGE_VIOLATION_ON_CALLED_DEVICE (9).

- Class of Service (COS) — The Class of Service for the callingDevice is never checked for the Make Call Service.
- Data Calls — Data calls cannot be originated via the Make Call Service.
- DCS — A call made by Make Call Service over a DCS network is treated as an off-PBX call.
- Display — If the callingDevice has a display set, the display will show the extension and name of the calledDevice, if the calledDevice is on-PBX, or the name of the trunk group, if the calledDevice is off-PBX. If the calledDevice is on-PBX, normal display interactions apply for calledDevice with displays.
- Forced Entry of Account Codes — Make Call Service request attempted to trunk groups with the Forced Entry of Account Codes feature assigned which is allowed. It is up to the user at the callingDevice to enter the account codes via the touch-tone pad. Account code may not be provided via the TSAPI. If the originator of such a call is logged into an adjunct-controlled split (and therefore has the voice set locked), such a user will be unable to enter the required codes and will eventually get denial treatment.
- Hot Line — A Make Call Service request made on behalf of a station that has the Hot Line feature administered will be denied.
- Last Number Dialed — The calledDevice in a Make Call Service request is the last number dialed for the calledDevice until the next call origination from the callingDevice. Therefore, the user can use the “last number dialed” button to originate a call to the destination provided in the last Make Call Service request.
- Logical Agents — The callingDevice may contain a logical agent’s login ID or a logical agent’s physical station. If a logical agent’s login ID is specified and the logical agent is logged in, the call is originated from the agent’s station extension associated with the agent’s login ID. If a logical agent’s login ID is specified and the logical agent is not logged in, the call is denied with error INVALID_CALLING_DEVICE.

If the calledDevice contains a logical agent’s login ID, the call is originated as if the call had been dialed from the callingDevice to the requested login ID. If the callingDevice and the calledDevice CORs permit, the call is treated as a direct agent call; otherwise, the call is treated as a personal call to the requested agent.

- Night Service — Make Call Service to splits in night service will go to night service.
- Personal Central Office Line (PCOL) — For a Make Call Service request originated at the PCOL call appearance of a primary extension, that user must be off-hook on the PCOL call appearance at the time the service is requested.
- PRI — An outgoing call over a PRI facility provides call feedback events from the network.

- Priority Calling — The user can originate a priority call by going off-hook, dialing the feature access code for priority calling, and requesting a Make Call Service.
- Send All Calls (SAC) — Make Call Service can be requested for a station (callingDevice) that has SAC activated. SAC has no effect on the callingDevice for the cstaMakeCall request.
- Single-Digit Dialing — Make Service request accepts single-digit dialing (for example, 0 for operator).
- Skill Hunt Groups — Make Call Service cannot have a skill hunt group extension as the callingDevice.
- Station Message Detail Recording (SMDR) — Calls originated by an application via the Make Call Service are marked with the condition code "B".
- Switch Operation — Once the call is successfully originated, the switch will not drop it regardless of outcome. The only exception is the denial outcome, which results in the intercept tone being played for 30 seconds after the call is disconnected. The originating station user or application drops cstaMakeCall calls either by going on-hook or via CSTA call control services. For example, if the application places a call to a busy destination, the originator will be busy until he/she normally drops or until the application sends a Clear Call or Clear Connection Service to drop the call.
- Terminating Extension Group (TEG) — Make Call Service requests cannot have the TEG group extension as the callingDevice. TEGs can only receive calls, not originate them.
- VDN — VDN cannot be the callingDevice of a Make Call Service, but it can be the calledDevice.
- VDN Destination — When the calledDevice is a VDN extension, vector processing rules apply.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaMakeCall() - Service Request

RetCode_t      cstaMakeCall (
    ACSHandle_t    acsHandle,
    InvokeID_t     invokeID,
    DeviceID_t     *callingDevice,
    DeviceID_t     *calledDevice,
    PrivateData_t  *privateData);

// CSTAMakeCallConfEvent - Service Response

typedef struct
{
    ACSHandle_t    acsHandle;
    EventClass_t   eventClass; // CSTACONFIRMATION
    EventType_t    eventType; // CSTA_HOLD_CALL_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t  eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
            union
            {
                CSTAMakeCallConfEvent_t    makeCall;
            } u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAMakeCallConfEvent_t {
    Nulltype    null;
} CSTAMakeCallConfEvent_t;
```

Private Data Version 6 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attV6MakeCall() - Service Request Private Data Setup
Function

RetCode_t    attV6MakeCall(
    ATTPrivateData_t*privateData,
    DeviceID_t    *split,        // NULL indicates not
specified
    ATTUserToUserInfo_t *userInfo); // NULL indicates not
// specified

typedef struct ATTPrivateData_t {
    char    vendor[32];
    ushort length;
    char    data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

#define ATT_MAX_USER_INFO 129
#define ATT_MAX_UII_SIZE 96
#define ATTV5_MAX_UII_SIZE 32

typedef struct ATTUserToUserInfo_t {
    ATTUIIProtocolType_t type;
    struct {
        short length; // 0 indicates UII not present
        unsigned char value[ATT_MAX_USER_INFO];
    } data;
} ATTUserToUserInfo_t;

typedef enum ATTUIIProtocolType_t {
    UII_NONE          = -1, // indicates not specified
    UII_USER_SPECIFIC = 0, // user-specific
    UII_IA5_ASCII     = 4, // null-terminated ascii
                        // character string
} ATTUIIProtocolType_t;
```

Private Data Version 6 Syntax (Continued)

```
// ATMakeCallConfEvent - Service Response Private
// Data (supported by private data version 5 and later only)

typedef struct
{
    ATTEventType_t eventType;// ATT_MAKE_CALL_CONF
    union
    {
        ATMakeCallConfEvent_t      makeCall;
    }u;
} ATTEvent_t;

typedef struct ATMakeCallConfEvent_t
{
    ATTUCID_t   ucid;
} ATMakeCallConfEvent_t;

typedef char ATTUCID_t[64];
```


Private Data Version 2-5 Syntax

```

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attMakeCall() - Service Request Private Data Setup Function

RetCode_t    attMakeCall(
    ATTPrivateData_t    *privateData,
    DeviceID_t          *split,        // NULL indicates not specified
    ATTUserToUserInfo_t *userInfo);   // NULL indicates not
                                        // specified

typedef struct ATTPrivateData_t {
    char    vendor[32];
    ushort length;
    char    data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef struct    ATTV5UserToUserInfo_t {
    ATTUIProtocolType_t type;
    struct {
        short length; // 0 indicates UUI not present
        unsigned char value[33];
    } data;
} ATTV5UserToUserInfo_t;

typedef enum ATTUIProtocolType_t {
    UUI_NONE          = -1,    // indicates not specified
    UUI_USER_SPECIFIC = 0,    // user-specific
    UUI_IA5_ASCII     = 4     // null-terminated ascii
                                // character string
} ATTUIProtocolType_t;

// ATTMakeCallConfEvent - Service Response Private
// Data (supported by private data version 5 and later only)

typedef struct
{
    ATTEventType_t    eventType; // ATT_MAKE_CALL_CONF
    union
    {
        {
            ATTMakeCallConfEvent_t    makeCall;
        }u;
    }
} ATTEvent_t;

typedef struct ATTMakeCallConfEvent_t
{
    ATTUCID_t    ucid;
} ATTMakeCallConfEvent_t;

typedef char ATTUCID_t[64];

```

Make Direct-Agent Call Service

Direction: Client to Switch

Function: *cstaMakeCall()*

Confirmation Event: *CSTAMakeCallConfEvent*

Private Data Function: *attV6DirectAgentCall()* (private data version 6),
attDirectAgentCall() (private data version 2, 3, 4, and 5)

Private Data Confirmation Event: *ATTMakeCallConfEvent*

Service Parameters: *callingDevice, calledDevice*

Private Parameters: *split, priorityCalling, userInfo*

Ack Parameters: *newCall*

Ack Private Parameters: *ucid*

Nak Parameter: *universalFailure*

Functional Description:

Make Direct-Agent Call Service is a special type of Make Call Service. Make Direct-Agent Call Service originates a call between two devices: a user station and an ACD agent logged into a specified split. The service attempts to create a new call and establish a connection with the originating device (*callingDevice*) first. The Direct-Agent Call service also provides a CSTA connection Identifier (*newCall*) that indicates the connection of the originating device in the *CSTAMakeCallConfEvent*.

This type of call may be used by applications whenever the application decides that the call originator should talk to a specific ACD agent. The application must specify the split extension (via database lookup) to which the *calledDevice* (ACD agent) is logged in. Direct-Agent calls can be tracked by Call Management Service (CMS) through the split measurements.

Service Parameters:

- callingDevice*** [mandatory] Must be a valid station extension or an AWOH station extension (for phantom calls).¹ This parameter may contain a logical agent's login ID (Logical Direct-Agent Call) or an agent's physical station extension. If the *callingDevice* contains a logical agent's login ID and the logical agent is logged in, the direct-agent call is originated from the agent's station. If the *callingDevice* contains a logical agent's login ID and the logical agent is not logged in, the direct-agent call is denied. The Logical Direct-Agent Call is only available when the Expert Agent Selection (EAS) feature is enabled on the G3 switch.
- calledDevice*** [mandatory] Must be a valid ACD agent extension. Agent at *calledDevice* must be logged in. If *calledDevice* is a logical agent's ID, it is already treated by DEFINITY as a direct agent call and, in this case, private data should not be used. Doing so would result in error INVALID_CSTA_DEVICE_IDENTIFIER (12).

1. For DEFINITY ECS switch software Release 6.3 and later, a call can be originated from an AWOH station or some group extensions (i.e., a plain [non-ACD] hunt group). This is termed a *phantom call*. Most calls that can be requested for a physical extension can also be requested for an AWOH station and the associated event will also be received. If the call is made on behalf of a group extension, this may not apply. For a detailed description of the phantom call switch feature, refer to CallVisor technical Reference (555-230-220).

Private Parameters:

- split*** [mandatory] Contains a valid split extension. Agent at calledDevice must be logged into this split.
- priorityCalling*** [mandatory] Specifies the priority of the call. Values are On (TRUE) or Off (FALSE). If On is selected, a priority call is attempted for an on-PBX calledDevice. Note that the G3 PBX does not permit priority calls to certain types of extensions (such as VDNs).

userInfo [optional] Contains user-to-user information. This parameter allows the application to associate caller information, up to 32 or 96 bytes, with a call. It may be a customer number, credit card number, alphanumeric digits, or a binary string.

It is propagated with the call. The switch sends the UUI in the Delivered Event Report to the application. A NULL indicates that this parameter is not present.

Prior to G3V8, the maximum length of userInfo was 32 bytes. Beginning with G3V8, the maximum length of userInfo was increased to 96 bytes.

⇒ NOTE:

An application using private data version 5 and earlier can only receive a maximum of 32-byte data for userInfo, regardless of the size data that is sent by the switch.

The following UUI protocol types are supported:

- UUI_NONE — There is no data provided in the data parameter.
- UUI_USER_SPECIFIC — The content of the data parameter is a binary string. The correct size (maximum of 32 or 96 bytes) of data must be specified in the size parameter.
- UUI_IA5_ASCII — The content of the data parameter must be a null-terminated IA5 (ASCII) character string. The correct size (maximum of 32 or 96 bytes excluding the null terminator) of data must be specified in the size parameter.

Ack Parameters:

newCall [mandatory] A connection identifier that indicates the connection between the originating device and the call. The newCall parameter contains the callID of the call and the station extension of the callingDevice.

Ack Private Parameters:

ucid [optional] Specifies the Universal Call ID (UCID) of newCall. The UCID is a unique call identifier across switches and the network. A valid UCID is a null-terminated ASCII character string. If there is no UCID associated with this call, the *ucid* contains the ATT_NULL_UCID (a 20-character string of all zeros). This parameter is supported by private data version 5 and later only.

Nak Parameter:

A Make Call request will be denied if the request fails before the call is attempted by the PBX.

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- **GENERIC_UNSPECIFIED (0)** The specified data provided for the *userInfo* parameter exceeds the maximum allowable size. Prior to G3V8, the maximum length of *userInfo* was 32 bytes. Beginning with G3V8, the maximum length of *userInfo* was increased to 96 bytes. See the description of the *userInfo* parameter.
- **GENERIC_UNSPECIFIED (0) (CS3/11, CS3/15)** Agent is not a member of the split or agent is not currently logged in split.
- **VALUE_OUT_OF_RANGE (3) (CS0/100, CS0/96)** The split contains an invalid value or invalid information element contents was detected.
- **INVALID_CALLING_DEVICE (5) (CS3/27)** The *callingDevice* is out of service or not administered correctly in the switch.
- **PRIVILEGE_VIOLATION_ON_CALLED_DEVICE (9) (CS0/21, CS0/52)** The COR check for completing the call failed. The call was attempted over a trunk that the originator has restricted from use.
- **INVALID_DESTINATION (14) (CS3/24)** The call was answered by an answering machine.
- **INVALID_OBJECT_STATE (22) (CS0/98)** Request (message) is incompatible with call state.

- **INVALID_CSTA_DEVICE_IDENTIFIER (12) (CS0/28)**
The split does not contain a valid hunt group extension. The callingDevice or calledDevice is an invalid station extension.
- **INVALID_OBJECT_TYPE (18) (CS0/58, CS3/80)** There is an incompatible bearer service for the originating or destination address (for example, the originator is administered as a data hotline station or the destination is a data station).

Call options are incompatible with this service.
- **GENERIC_STATE_INCOMPATIBILITY (21) (CS0/18)**
The originator does not go off-hook within five seconds after originating the call and cannot be forced off-hook.
- **RESOURCE_BUSY (33) (CS0/17)** The user is busy on another call and cannot originate this call. The switch is busy with another CSTA request. This can happen when two G3PDs are issuing requests (for example, Hold Call, Retrieve Call, Clear Connection, Make Call, etc.) to the same device.
- **GENERIC_SUBSCRIBED_RESOURCE_AVAILABILITY (41) (CS0/50)** Service or option not subscribed/provisioned (AMD must be enabled).

Detailed Information:

See “Detailed Information:” in the “Make Call Service” section in this chapter.

- **Display** — If the calledDevice has a display set, it will show the specified split’s name and extension. If the destination ACD agent has a display, it will show the name of the originator and the name of the specified split.
- **Logical Agents** — The callingDevice may contain a logical agent’s login ID or a logical agent’s physical station. If a logical agent’s login ID is specified and the logical agent is logged in, the call originates from the agent’s station extension associated with the agent’s login ID. If a logical agent’s login ID is specified and the logical agent is not logged in, the call is denied with the error **INVALID_CALLING_DEVICE**.

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaMakeCall() - Service Request

RetCode_t      cstaMakeCall (
    ACSHandle_t   acsHandle,
    InvokeID_t    invokeID,
    DeviceID_t    *callingDevice,
    DeviceID_t    *calledDevice,
    PrivateData_t *privateData);

// CSTAMakeCallConfEvent - Service Response

typedef struct
{
    ACSHandle_t   acsHandle;
    EventClass_t  eventClass; // CSTACONFIRMATION
    EventType_t  eventType; // CSTA_MAKE_CALL_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t  eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
            union
            {
                CSTAMakeCallConfEvent_t  makeCall;
            } u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAMakeCallConfEvent_t
{
    ConnectionID_t newCall; // devIDType = STATIC_ID
} CSTAMakeCallConfEvent_t;

```

Private Data Version 6 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attV6DirectAgentCall() - Service Request Private Data
// Setup Function

RetCode_t attV6DirectAgentCall(
    ATTPrivateData_t*privateData,
    DeviceID_t      *split,      // mandatory
                                // NULL indicates not
specified
    Boolean         priorityCalling;// TRUE = On, FALSE = Off
    ATTUserToUserInfo_t *userInfo); // NULL indicates not
                                // specified

typedef struct ATTPrivateData_t {
    char    vendor[32];
    ushort  length;
    char    data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

#define ATT_MAX_USER_INFO 129
#define ATT_MAX_UII_SIZE 96
#define ATTV5_MAX_UII_SIZE 32

typedef struct ATTUserToUserInfo_t {
    ATTUIIProtocolType_t type;
    struct {
        short length;      // 0 indicates UII not present
        unsigned char value[ATT_MAX_USER_INFO];
    } data;
} ATTUserToUserInfo_t;

typedef enum ATTUIIProtocolType_t {
    UII_NONE          = -1, // indicates not specified
    UII_USER_SPECIFIC = 0, // user-specific
    UII_IA5_ASCII     = 4, // null-terminated ascii
                                // character string
} ATTUIIProtocolType_t;
```


Private Data Version 6 Syntax (Continued)

```
// ATMakeCallConfEvent - Service Response Private
// Data (supported by private data version 5 and later only)

typedef struct
{
    ATTEventType_t eventType;// ATT_MAKE_CALL_CONF
    union
    {
        ATMakeCallConfEvent_t      makeCall;
    }u;
} ATTEvent_t;

typedef struct ATMakeCallConfEvent_t
{
    ATTUCID_t   ucid;
} ATMakeCallConfEvent_t;

typedef char ATTUCID_t[64];
```

Private Data Version 2-5 Syntax

```

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attDirectAgentCall() - Service Request Private Data
// Setup Function

RetCode_t attDirectAgentCall(
    ATTPrivateData_t *privateData,
    DeviceID_t *split, // mandatory
                        // NULL indicates not specified
    Boolean priorityCalling; // TRUE = On, FALSE = Off
    ATTUserToUserInfo_t *userInfo); // NULL indicates not
                                    // specified

typedef struct ATTPrivateData_t {
    char vendor[32];
    ushort length;
    char data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef struct ATTV5UserToUserInfo_t {
    ATTUUIProtocolType_t type;
    struct {
        short length; // 0 indicates UUI not present
        unsigned char value[33];
    } data;
} ATTV5UserToUserInfo_t;

typedef enum ATTUUIProtocolType_t {
    UUI_NONE = -1, // indicates not specified
    UUI_USER_SPECIFIC = 0, // user-specific
    UUI_IA5_ASCII = 4 // null-terminated ascii
                        // character string
} ATTUUIProtocolType_t;

// ATTMakeCallConfEvent - Service Response Private
// Data (supported by private data version 5 and later only)

typedef struct
{
    ATTEventType_t eventType; // ATT_MAKE_CALL_CONF
    union
    {
        {
            ATTMakeCallConfEvent_t makeCall;
        }u;
    }
} ATTEvent_t;

typedef struct ATTMakeCallConfEvent_t
{
    ATTUCID_t ucid;
} ATTMakeCallConfEvent_t;

typedef char ATTUCID_t[64];

```

Make Predictive Call Service

Direction: Client to Switch

Function: *cstaMakePredictiveCall()*

Confirmation Event: *CSTAMakePredictiveCallConfEvent*

Private Data Function: *attV6MakePredictiveCall()* (private data version 6),
attMakePredictiveCall() (private data version 2, 3, 4, and 5)

Private Data Confirmation Event: *ATTMakePredictiveCallConfEvent*

Service Parameters: *callingDevice, calledDevice, allocationState*

Private Parameters: *priorityCalling, maxRings, answerTreat, destRoute, userInfo*

Ack Parameters: *newCall*

Ack Private Parameters: *ucid*

Nak Parameter: *universalFailure*

Functional Description:

The Make Predictive Call Service originates a Switch-Classified call between two devices. The service attempts to create a new call and establish a connection with the terminating (called) device first. The Make Predictive Call service also provides a CSTA Connection Identifier that indicates the connection of the terminating device. The call will be dropped if the call is not answered after the maximum ring cycle has expired. When a G3 is administered to return a classification, the classification appears in the Established event.

Predictive dial calls cannot use TAC dialing to either access trunks or to make outbound calls — TAC dialing will be blocked by the DEFINITY switch.

Service Parameters:

- callingDevice** [mandatory] Must be a valid local extension number associated with an ACD split, hunt group, or announcement, a VDN in an EAS environment, or an AWOH station extension (for phantom calls).¹
- calledDevice** [mandatory] Must be a valid on-PBX extension or off-PBX number. On-PBX extension must be a station extension. The calledDevice may include ARS/AAR information for off-PBX numbers. Authorization Codes and Force Entry of Account Codes can be specified with the calledDevice as if they were entered from the voice terminal using the keypad.
- allocationState** [optional — partially supported] Specifies the condition in which the call attempts to connect to the caller (callingDevice). Only AS_CALL_ESTABLISHED is supported, meaning that G3 PBX will attempt to connect the call to the callingDevice if connected state is determined at the calledDevice. If AS_CALL_DELIVERED is specified, it will be ignored and default to AS_CALL_ESTABLISHED.

1. For DEFINITY ECS switch software Release 6.3 and later, a call can be originated from an AWOH station or some group extensions (i.e., a plain [non-ACD] hunt group). This is termed a *phantom call*. Most calls that can be requested for a physical extension can also be requested for an AWOH station and the associated event will also be received. If the call is made on behalf of a group extension, this may not apply. For a detailed description of the phantom call switch feature, refer to CallVisor technical Reference (555-230-220).

Private Parameters:

- priorityCalling** [mandatory] Specifies the priority of the call. Values are On (TRUE) or Off (FALSE). If On is selected, a priority call is attempted for an on-PBX calledDevice. Note that the G3 PBX does not permit priority calls to certain types of extensions (such as VDNs).
- maxRings** [optional] Specifies the number of rings that are allowed before classifying the call as no answer. The minimum is two; the maximum is 15. If an out-of-range value is specified, it defaults to 10.
- answerTreat** [mandatory] Specifies the call treatment when an answering machine is detected.

- AT_NONE — Treatment follows the switch answering machine detection administration.
- AT_DROP — Drops the call if an answering machine is detected.
- AT_CONNECT — Connects the call if an answering machine is detected.
- AT_NO_TREATMENT — Indicates that no answering machine treatment is specified.

destRoute [optional] Specifies the TAC/ARS/AAR information for off-PBX destinations if the information is not included in the calledDevice. A NULL indicates that this parameter is not specified.

userInfo [optional] Contains user-to-user information. This parameter allows the application to associate caller information, up to 32 or 96 bytes, with a call. It may be a customer number, credit card number, alphanumeric digits, or a binary string.

It is propagated with the call whether the call is made to a destination on the local switch or to a destination on a remote switch over PRI trunks. The switch sends the UUI in the ISDN SETUP message over the PRI trunk to establish the call. The local and the remote switch include the UUI in the Delivered Event Report and in the cstaRouteRequestEvent to the application. A NULL indicates that this parameter is not present.

Prior to G3V8, the maximum length of userInfo was 32 bytes. Beginning with G3V8, the maximum length of userInfo was increased to 96 bytes.

⇒ NOTE:

An application using private data version 5 and earlier can only receive a maximum of 32-byte data for userInfo, regardless of the size data that is sent by the switch.

The following UUI protocol types are supported:

- UUI_NONE — There is no data provided in the data parameter.
- UUI_USER_SPECIFIC — The content of the data parameter is a binary string. The correct size (maximum of 32 or 96 bytes) of data must be specified in the size parameter.
- UUI_IA5_ASCII — The content of the data parameter must be a null-terminated IA5 (ASCII) character string. The correct size (maximum of 32 or 96 bytes excluding the null terminator) of data must be specified in the size parameter.

Ack Parameters:

newCall [mandatory] A connection identifier that indicates the connection between the originating device and the call. The *newCall* parameter contains the callID of the call and the station extension of the *callingDevice*.

Ack Private Parameters:

ucid [optional] Specifies the Universal Call ID (UCID) of *newCall*. The UCID is a unique call identifier across switches and the network. A valid UCID is a null-terminated ASCII character string. If there is no UCID associated with this call, the *ucid* contains the ATT_NULL_UCID (a 20-character string of all zeros). This parameter is supported by private data version 5 and later only.

Nak Parameter:

A Make Call request will be denied if the request fails before the call is attempted by the PBX.

universalFailure If the request is not successful, the application will receive a *CSTAUniversalFailureConfEvent*. The error parameter in this event may contain the following error values, or one of the error values described in the “*CSTAUniversalFailureConfEvent*” section in Chapter 3:

- **GENERIC_UNSPECIFIED (0)** The specified data provided for the *userInfo* parameter exceeds the maximum allowable size. Prior to G3V8, the maximum length of *userInfo* was 32 bytes. Beginning with G3V8, the maximum length of *userInfo* was increased to 96 bytes. See the description of the *userInfo* parameter.
- **VALUE_OUT_OF_RANGE (3) (CS0/100, CS0/96)** Invalid information element contents was detected.
- **INVALID_CALLING_DEVICE (5) (CS3/27)** The *callingDevice* is out of service or not administered correctly in the switch.
- **PRIVILEGE_VIOLATION_ON_CALLED_DEVICE (9) (CS0/21, CS0/52)** Attempted to use a Trunk Access Code (TAC) to access a PRI trunk (only AAR/ARS feature access codes may be used to place a switch-classified call over a PRI trunk). The COR check for completing the call failed. The call was attempted over a trunk that the originator has restricted from use.

- INVALID_CSTA_DEVICE_IDENTIFIER (12) (CS0/28)
The callingDevice is neither a split nor an announcement extension.
- INVALID_OBJECT_TYPE (18) (CS0/58, CS3/80) There is incompatible bearer service for the originating or destination address. For example, the originator is administered as a data hotline station or the destination is a data station. Call options are incompatible with this service.
- INVALID_OBJECT_STATE (22) (CS0/98) Request (message) is incompatible with the call state.
- GENERIC_SYSTEM_RESOURCE_AVAILABILITY (31) (CS3/22)
One of the following conditions exists when switch attempted to make the call:
 - No Call classifier available
 - No time slot available
 - No trunk available
 - Queue full
- GENERIC_SUBSCRIBED_RESOURCE_AVAILABILITY (41) (CS0/50) Service or option not subscribed/provisioned. Answer Machine Detection is requested, but AMD is not enabled on the switch.

Detailed Information:

- NT Version 4.2/NetWare Release 2.22d and earlier: Prior to Release 2.22e, the CSTAMakePredictiveCallConfEvent is not sent to the application until the destination answers the call and the switch connects the call to the calling device. When the application receives the CSTAMakePredictiveCallConfEvent, it is too late to monitor the call and receive the events for connecting the destination. This has been corrected in Release 2.22e.
- NT Version 4.3/NetWare Release 2.22e and later: The CSTAMakePredictiveCallConfEvent is sent to the application immediately after the switch accepts the CSTAMakePredictiveCall request and attempts to call the destination. The application receives a call ID in the CSTAMakePredictiveCallConfEvent. The application can monitor the outbound call and receives events of the call when the switch tries to connect the destination. When the outbound call is monitored, the call ID in the reported events remains unchanged when the destination answers and when the switch connects the calling device (normally this is a VDN or an ACD Split); that is, the call ID remains unchanged until the call is conferenced or transferred.

⇒ NOTE:

If the calling device (the VDN or the ACD Split) is monitored by CSTAMonitorCallsViaDevice (or the agent receives a call monitored by CSTAMonitorDevice), when the CSTAMakePredictiveCall reaches the VDN/ACD Split (or when an agent connects to the call), the call ID in the reported events from CSTAMonitorCallsViaDevice (or CSTAMonitorDevice) is different from the one reported in the CSTAMakePredictiveCallConfEvent and the CSTAMonitorCall (using the call ID reported in the CSTAMakePredictiveCallConfEvent). In other words, there are two call IDs associated with the same CSTAMakePredictiveCall and they are reported in events of their respective monitorings, however, both call IDs can be used for CSTAMonitorCall and the call IDs will remain the same during their respective monitorings until the call is conferenced or transferred.

⇒ NOTE:

The switch assigns a call ID to the outbound call when it connects to the destination and a different call ID to the inbound call when it arrives at the VDN/ACD Split. If the outbound call is monitored and the VDN/ACD Split (or the agent's station) is also monitored, an application cannot determine that the two unique call IDs are actually associated with the same CSTAMakePredictiveCall. Other methods must be used to determine that these two call IDs are associated with the same call. One way to do this is to use the called device parameter, if it is unique among all calls. Another way is to use the UUI parameter. The application can send a unique ID in the UUI with the CSTAMakePredictiveCall and this ID is reported in the events of the CSTAMonitorCallsViaDevice for the VDN/ACD Split (or the CSTAMonitorDevice for the agent's station), if the UUI is not used for another purpose.

⇒ NOTE:

The callingDevice and the calledDevice in the event reports resulting from the outbound call monitored by CSTAMonitorCall (using the call ID reported in the CSTAMakePredictiveCallConfEvent) are the same as those specified in the CSTAMakePredictiveCall request. However, this is different from the callingDevice and calledDevice in the events reported from the CSTAMonitorCallsViaDevice of the VDN/ACD Split or the CSTAMonitorDevice of the agent station. These monitors have an inbound call view instead of an outbound call view. Thus, the callingDevice is the calledDevice specified in the CSTAMakePredictiveCall request. The calledDevice is the callingDevice specified in the CSTAMakePredictiveCall request.

⇒ NOTE:

If a client application wants to receive events for answering machine detection, the client application should establish a monitorCall, after the application receives a confirmation for the makePredictiveCall.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaMakePredictiveCall() - Service Request

RetCode_t      cstaMakePredictiveCall (
    ACSHandle_t      acsHandle,
    InvokeID_t      invokeID,
    DeviceID_t      *callingDevice,
    DeviceID_t      *calledDevice,
    AllocationState_t allocationState,
    PrivateData_t    *privateData);

// CSTAMakePredictiveCallConfEvent - Service Response

typedef struct
{
    ACSHandle_t      acsHandle;
    EventClass_t     eventClass; // CSTACONFIRMATION
    EventType_t      eventType; // CSTA_MAKE_PREDICTIVE_CALL_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            InvokeID_t invokeID;
            union
            {
                CSTAMakePredictiveCallConfEvent_t makePredictiveCall;
            } u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAMakePredictiveCallConfEvent_t
{
    ConnectionID_t newCall; // devIDType = STATIC_ID
} CSTAMakePredictiveCallConfEvent_t;
```

Private Data Version 6 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attV6MakePredictiveCall() - Service Request Private Data
// Setup Function

RetCode_t attV6MakePredictiveCall(
    ATTPrivateData_t*privateData,
    Boolean          priorityCalling;//TRUE = On, FALSE = Off
    short           maxRings,      // less than 2 or greater 15
                                   // are treated as not
                                   // specified
    ATTAnswerTreat_tanswerTreat,  // AT_NONE, AT_DROP, or
                                   // AT_CONNECT
    DeviceID_t      *destRoute,    // NULL = not specified
    ATTUserToUserInfo_t *userInfo); // NULL = not specified

typedef struct ATTPrivateData_t {
    char    vendor[32];
    ushort length;
    char    data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

#define ATT_MAX_USER_INFO 129
#define ATT_MAX_UII_SIZE 96
#define ATTV5_MAX_UII_SIZE 32

typedef struct ATTUserToUserInfo_t {
    ATTUIIProtocolType_t type;
    struct {
        short length; // 0 indicates UII not present
        unsigned char value[ATT_MAX_USER_INFO];
    } data;
} ATTUserToUserInfo_t;
```

Private Data Version 6 Syntax (Continued)

```
typedef enum ATTUUIProtocolType_t {
    UUI_NONE          = -1, // indicates not specified
    UUI_USER_SPECIFIC = 0, // user-specific
    UUI_IA5_ASCII     = 4, // null-terminated ascii
                        // character string
} ATTUUIProtocolType_t;

typedef enum ATTAnswerTreat_t {
    AT_NO_TREATMENT= 0, // indicates treatment not specified

    AT_NONE      = 1, // treatment follows machine
instruct
    AT_DROP      = 2, // drop call if machine detected
    AT_CONNECT   = 3, // connect call if machine detected

} ATTAnswerTreat_t;

// ATTMakePredictiveCallConfEvent - Service Response
// Private Data
(supported by private data version 5 and later only)

typedef struct
{
    ATTEventType_t eventType;
                                // ATT_MAKE_PREDICTIVE_CALL_CONF
    union
    {
        ATTMakePredictiveCallConfEvent_tmakePredictiveCall;
    }u;
} ATTEvent_t;

typedef struct ATTMakePredictiveCallConfEvent_t
{
    ATTUCID_t ucid;
} ATTMakePredictiveCallConfEvent_t;

typedef char ATTUCID_t[64];
```

Private Data Version 2-5 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attMakePredictiveCall() - Service Request Private Data
// Setup Function

RetCode_t attMakePredictiveCall(
    ATTPrivateData_t*privateData,
    Boolean          priorityCalling;//TRUE = On, FALSE = Off
    short           maxRings,      // less than 2 or greater 15
                                   // are treated as not
                                   // specified
    ATTAnswerTreat_tanswerTreat,  // AT_NONE, AT_DROP, or
                                   // AT_CONNECT
    DeviceID_t      *destRoute,   // NULL = not specified
    ATTUserToUserInfo_t *userInfo); // NULL = not specified

typedef struct ATTPrivateData_t {
    char    vendor[32];
    ushort length;
    char    data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef struct ATTV5UserToUserInfo_t {
    ATTUIProtocolType_t type;
    struct {
        short length; // 0 indicates UII not present
        unsigned char value[33];
    } data;
} ATTV5UserToUserInfo_t;
```

Private Data Version 2-5 Syntax (Continued)

```
typedef enum ATTUUIProtocolType_t {
    UUI_NONE          = -1, // indicates not specified
    UUI_USER_SPECIFIC = 0, // user-specific
    UUI_IA5_ASCII     = 4, // null-terminated ascii
                        // character string
} ATTUUIProtocolType_t;

typedef enum ATTAnswerTreat_t {
    AT_NO_TREATMENT= 0, // indicates treatment not specified

    AT_NONE      = 1, // treatment follows machine
instruct
    AT_DROP      = 2, // drop call if machine detected
    AT_CONNECT   = 3, // connect call if machine detected

} ATTAnswerTreat_t;

// ATTMakePredictiveCallConfEvent - Service Response
// Private Data
(supported by private data version 5 and later only)

typedef struct
{
    ATTEventType_t eventType;
                                // ATT_MAKE_PREDICTIVE_CALL_CONF
    union
    {
        ATTMakePredictiveCallConfEvent_tmakePredictiveCall;
    }u;
} ATTEvent_t;

typedef struct ATTMakePredictiveCallConfEvent_t
{
    ATTUCID_t ucid;
} ATTMakePredictiveCallConfEvent_t;

typedef char ATTUCID_t[64];
```

Make Supervisor-Assist Call Service

Direction: Client to Switch

Function: *cstaMakeCall()*

Confirmation Event: *CSTAMakeCallConfEvent*

Private Data Function: *attV6SupervisorAssistCall()* (private data version 6),
attSupervisorAssistCall() (private data version 2, 3, 4, and 5)

Private Data Confirmation Event: *ATTMakeCallConfEvent*

Service Parameters: *callingDevice, calledDevice*

Private Parameters: *split, userInfo*

Ack Parameters: *newCall*

Ack Private Parameters: *ucid*

Nak Parameter: *universalFailure*

Functional Description:

Make Supervisor-Assist Call Service is a special type of Make Call Service. This service originates a call between two devices: an ACD agent's extension and another station extension (typically a supervisor). The service attempts to create a new call and establish a connection with the originating (calling) device first. The Supervisor-Assist Call service also provides a CSTA Connection Identifier that indicates the connection of the originating device.

A call of this type is measured by CMS as a supervisor-assist call and is always a priority call.

This type of call is used by the application whenever an agent wants to consult with the supervisor. The agent must be logged into the specified ACD split to use this service.

Service Parameters:

- callingDevice*** [mandatory] Must be a valid ACD agent extension or an AWOH station extension (for phantom calls).¹ Agent must be logged in.
- calledDevice*** [mandatory] Must be valid on-PBX station extension (excluding VDNs, splits, off-PBX DCS and UDP extensions).

-
1. For DEFINITY ECS switch software Release 6.3 and later, a call can be originated from an AWOH station or some group extensions (i.e., a plain [non-ACD] hunt group). This is termed a *phantom call*. Most calls that can be requested for a physical extension can also be requested for an AWOH station and the associated event will also be received. If the call is made on behalf of a group extension, this may not apply. For a detailed description of the phantom call switch feature, refer to CallVisor technical Reference (555-230-220).

Private Parameters:

split [mandatory] Specifies the ACD agent's split extension. The agent at callingDevice must be logged into this split.

userInfo [optional] Contains user-to-user information. This parameter allows the application to associate caller information, up to 32 or 96 bytes, with a call. It may be a customer number, credit card number, alphanumeric digits, or a binary string. It is propagated with the call. The switch sends the UUI in the Delivered Event Report to the application. A NULL indicates that this parameter is not present.

Prior to G3V8, the maximum length of userInfo was 32 bytes. Beginning with G3V8, the maximum length of userInfo was increased to 96 bytes.

NOTE:

An application using private data version 5 and earlier can only receive a maximum of 32-byte data for userInfo, regardless of the size data that is sent by the switch.

The following UUI protocol types are supported:

- UUI_NONE — There is no data provided in the data parameter.
- UUI_USER_SPECIFIC — The content of the data parameter is a binary string. The correct size (maximum of 32 or 96 bytes) of data must be specified in the size parameter.
- UUI_IA5_ASCII — The content of the data parameter must be a null-terminated IA5 (ASCII) character string. The correct size (maximum of 32 or 96 bytes excluding the null terminator) of data must be specified in the size parameter.

Ack Parameters:

newCall [mandatory] A connection identifier that indicates the connection between the originating device and the call. The newCall parameter contains the callID of the call and the station extension of the callingDevice.

Ack Private Parameters:

ucid [optional] Specifies the Universal Call ID (UCID) of newCall. The UCID is a unique call identifier across switches and the network. A valid UCID is a null-terminated ASCII character string. If there is no UCID associated with this call, the *ucid* contains the ATT_NULL_UCID (a 20-character string of all zeros). This parameter is supported by private data version 5 and later only.

Nak Parameter:

A Make Call request will be denied if the request fails before the call is attempted by the PBX.

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- **GENERIC_UNSPECIFIED (0)** The specified data provided for the *userInfo* parameter exceeds the maximum allowable size. Prior to G3V8, the maximum length of *userInfo* was 32 bytes. Beginning with G3V8, the maximum length of *userInfo* was increased to 96 bytes. See the description of the *userInfo* parameter.
- **GENERIC_UNSPECIFIED (0) (CS3/11, CS3/15)** The agent is not a member of the split or the agent is not currently logged into the split.
- **VALUE_OUT_OF_RANGE (3) (CS0/100, CS0/96)** The split contains an invalid value or invalid information element contents was detected.
- **INVALID_CALLING_DEVICE (5) (CS3/27)** The *callingDevice* is out of service or not administered correctly in the switch.
- **PRIVILEGE_VIOLATION_ON_CALLED_DEVICE (9) (CS0/21, CS0/52)** The COR check for completing the call failed. The call was attempted over a trunk that the originator has restricted from use.
- **INVALID_DESTINATION (14) (CS3/24)** The call was answered by an answering machine.
- **INVALID_CSTA_DEVICE_IDENTIFIER (12) (CS0/28)** The split does not contain a valid hunt group extension. The *callingDevice* or *calledDevice* is an invalid station extension.

- INVALID_OBJECT_TYPE (18) (CS0/58, CS3/80) There is incompatible bearer service for the originating or destination address. For example, the originator is administered as a data hotline station or the destination is a data station. Call options are incompatible with this service.
- GENERIC_STATE_INCOMPATIBILITY (21) (CS0/18) The originator does not go off-hook within five seconds after originating the call and cannot be forced off-hook.
- INVALID_OBJECT_STATE (22) (CS0/98) Request (message) is incompatible with call state.
- RESOURCE_BUSY (33) (CS0/17) The user is busy on another call and cannot originate this call. The switch is busy with another CSTA request. This can happen when two G3PDs are issuing requests (for example, Hold Call, Retrieve Call, Clear Connection, Make Call, etc.) to the same device.
- GENERIC_SUBSCRIBED_RESOURCE_AVAILABILITY (41) (CS0/50) Service or option not subscribed/provisioned (AMD must be enabled).

Detailed Information:

See “Detailed Information:” in the “Make Call Service” section in this chapter.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaMakeCall() - Service Request

RetCode_t      cstaMakeCall (
    ACSHandle_t   acsHandle,
    InvokeID_t    invokeID,
    DeviceID_t    *callingDevice,
    DeviceID_t    *calledDevice,
    PrivateData_t *privateData);

// CSTAMakeCallConfEvent - Service Response

typedef struct
{
    ACSHandle_t   acsHandle;
    EventClass_t  eventClass; // CSTACONFIRMATION
    EventType_t   eventType; // CSTA_MAKE_CALL_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t  eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
            union
            {
                CSTAMakeCallConfEvent_t  makeCall;
            } u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAMakeCallConfEvent_t
{
    ConnectionID_t newCall; // devIDType = STATIC_ID
} CSTAMakeCallConfEvent_t;
```

Private Data Version 6 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attV6SupervisorAssistCall() - Service Request Private Data
// Setup Function

RetCode_t attV6SupervisorAssistCall(
    ATTPrivateData_t*privateData,
    DeviceID_t      *split,      // mandatory
                                // NULL indicates not
specified
    ATTUserToUserInfo_t *userInfo); // NULL indicates not
                                // specified

typedef struct ATTPrivateData_t {
    char    vendor[32];
    ushort length;
    char    data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

#define ATT_MAX_USER_INFO 129
#define ATT_MAX_UII_SIZE 96
#define ATTV5_MAX_UII_SIZE 32

typedef struct ATTUserToUserInfo_t {
    ATTUIIProtocolType_t type;
    struct {
        short length;      // 0 indicates UII not present
        unsigned char value[ATT_MAX_USER_INFO];
    } data;
} ATTUserToUserInfo_t;

typedef enum ATTUIIProtocolType_t {
    UII_NONE          = -1, // indicates not specified
    UII_USER_SPECIFIC = 0, // user-specific
    UII_IA5_ASCII     = 4   // null-terminated ascii
                        // character string
} ATTUIIProtocolType_t;
```

Private Data Version 6 Syntax (Continued)

```
// ATMakeCallConfEvent - Service Response Private
// Data (supported by private data version 5 and later only)

typedef struct
{
    ATTEventType_t eventType;// ATT_MAKE_CALL_CONF
    union
    {
        ATMakeCallConfEvent_t      makeCall;
    }u;
} ATTEvent_t;

typedef struct ATMakeCallConfEvent_t
{
    ATTUCID_t   ucid;
} ATMakeCallConfEvent_t;

typedef char ATTUCID_t[64];
```

Private Data Version 2-5 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attSupervisorAssistCall() - Service Request Private Data
// Setup Function

RetCode_t attSupervisorAssistCall(
    ATTPrivateData_t*privateData,
    DeviceID_t      *split,      // mandatory
                                // NULL indicates not
specified
    ATTUserToUserInfo_t *userInfo); // NULL indicates not
                                // specified

typedef struct ATTPrivateData_t {
    char    vendor[32];
    ushort  length;
    char    data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef struct ATTV5UserToUserInfo_t {
    ATTUUIProtocolType_t type;
    struct {
        short length;      // 0 indicates UUI not present
        unsigned char value[33];
    } data;
} ATTV5UserToUserInfo_t;

typedef enum ATTUUIProtocolType_t {
    UUI_NONE          = -1, // indicates not specified
    UUI_USER_SPECIFIC = 0, // user-specific
    UUI_IA5_ASCII     = 4, // null-terminated ascii
                                // character string
} ATTUUIProtocolType_t;
```

Private Data Version 2-5 Syntax (Continued)

```
// ATMakeCallConfEvent - Service Response Private
// Data (supported by private data version 5 and later only)

typedef struct
{
    ATTEventType_t eventType;// ATT_MAKE_CALL_CONF
    union
    {
        ATMakeCallConfEvent_t      makeCall;
    }u;
} ATTEvent_t;

typedef struct ATMakeCallConfEvent_t
{
    ATTUCID_t ucid;
} ATMakeCallConfEvent_t;

typedef char ATTUCID_t[64];
```


Pickup Call Service

Direction: Client to Switch
Function: *cstaPickupCall ()*
Confirmation Event: *CSTAPickupCallConfEvent*
Service Parameters: *deflectCall, calledDevice*
Ack Parameters: *noData*
Nak Parameter: *universalFailure*

Functional Description:

This service takes an alerting call at a device to another on-PBX device (within a DCS environment). The call at the alerting device is dropped after a successful redirection. An application may take an alerting call (at different devices) to another device any number of times until the call is answered or dropped by the caller.

The service request is positively acknowledged, if the call has successfully taken to another device.

If the service request is negatively acknowledged, the call remains at the alerting device and the calledDevice is not involved in the call.

Service Parameters:

deflectCall [mandatory] Specifies the connectionID of the call that is to be taken to another destination. The call must be in alerting state at the device. The device must be a valid voice station extension.

calledDevice [mandatory] Specifies the destination of the call. The destination must be an on-PBX endpoint. The calledDevice may be stations, queues, announcements, VDNs, or logical agent extension. Note that the calledDevice can be a device within a DCS environment.

Ack Parameter:

noData None for this service.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- PRIVILEGE_VIOLATION_ON_CALLED_DEVICE (9) (CS3/42)
 - Attempted to take a call back to the call originator or to the alerting device itself.
 - Attempted to take a call on the calledDevice of a cstaMakePredictiveCall.
- INVALID_OBJECT_STATE (22) (3/63)
 - An invalid callID or device identifier is specified in deflectCall.
 - The deflectCall is not at alerting state.
 - Attempted to take the call while in vector processing.
- INVALID_DESTINATION (14) (CS3/43) The request may fail because of one of the following:
 - Invalid destination specified
 - Toll restrictions on destination
 - COR restrictions on destination
 - Destination is remote access extension
 - Call origination restriction on the redirecting device
 - Call is in vector processing

Issue 1 — December 2001

- RESOURCE_BUSY (33) (CS0/17) The calledDevice is busy.
- GENERIC_SUBSCRIBED_RESOURCE_AVAILABILITY (41) (CS0/50) This service is requested on a G3 PBX administered as a release earlier than G3V4.
- GENERIC_OPERATION (1) (CS0/111) This service is requested on a queued call or protocol error in the request.

Detailed Information:

- Administration Without Hardware — A call cannot be redirected from/to an AWOH station. However, if the AWOH station is forwarded to a real physical station, the call can be redirected from/to such a station, if it is being alerted.
- Attendants — Calls on attendants cannot be redirected.
- Auto Call Back — ACB calls cannot be redirected by the cstaDeflectCall service from the call originator.
- Bridged Call Appearance — A call may be redirected away from a primary extension or from a bridged station. When that happens, the call is redirected away from the primary and all bridged stations.
- Call Forwarding, Cover All, Send All Call — Call redirection to a station with Call Forwarding/Cover All/Send All Call active can be picked up.
- Call Waiting — A call may be redirected while waiting at a busy analog set.
- cstaDeflectCall — The cstaPickupCall Service is similar to the cstaDeflectCall service, except that the calledDevice must be an on-PBX device. Note that the calledDevice can be a device within a DCS environment.
- Deflect From Queue — This service will not redirect a call from a queue to a new destination.
- Delivered Event — If the calling device or call is monitored, an application subsequently receives Delivered (or Network Reached) Event when redirection succeeds.
- Diverted Event — If the redirecting device is monitored by a cstaMonitorDevice or the call is monitored by a cstaMonitorCallsViaDevice, it will receive a Diverted Event when the call is successfully redirected, but there will be no Diverted Event for a cstaMonitorCall association.
- Loop Back — A call cannot be redirected back to call originator or to the redirecting device itself.

- **Off-PBX Destination** — If the call is redirected to an off-PBX destination, the caller will hear call progress tones. Some conditions (for example, trunk not available) may prevent the call from being placed. The call is nevertheless routed in those cases, and the caller receives busy or reorder treatment. An application may subsequently receive Failed, Call Cleared, Connection Cleared Events if redirection fails.

If trunk-to-trunk transfer is disallowed by the switch administration, redirection of an incoming trunk call to an off-PBX destination will fail.

- **Priority and Forwarded Calls** — Priority and forwarded calls are allowed to be redirected with `cstaDeflectCall`.
- **Service Availability** — This service is only available on a G3 PBX with G3V4 or later software.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaPickupCall() - Service Request

RetCode_t      cstaPickupCall (
    ACSHandle_t   acsHandle,
    InvokeID_t    invokeID,
    ConnectionID_t *deflectCall,
    DeviceID_t    *calledDevice,
    PrivateData_t *privateData);

// CSTAPickupCallConfEvent - Service Response

typedef struct
{
    ACSHandle_t   acsHandle;
    EventClass_t  eventClass; // CSTACONFIRMATION
    EventType_t   eventType; // CSTA_PICKUP_CALL_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t  eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
            union
            {
                CSTAPickupCallConfEvent_t  pickupCall;
            } u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAPickupCallConfEvent_t {
    Nulltype  null;
} CSTAPickupCallConfEvent_t;
```

Reconnect Call Service

Direction: Client to Switch

Function: `cstaReconnectCall()`

Confirmation Event: `CSTARReconnectCallConfEvent`

Private Data Function: `attV6ReconnectCall()` (private data version 6),
`attReconnectCall()` (private data version 2, 3, 4, and 5)

Service Parameters: `activeCall`, `heldCall`

Private Parameters: `dropResource`, `userInfo`

Ack Parameters: `noData`

Nak Parameter: `universalFailure`

Functional Description:

The Reconnect Call Service allows a client to disconnect (drop) an existing connection from a call and then reconnect a previously held connection or answer an alerting (or bridged) call at the same device. It provides the compound action of the Clear Connection Service followed by a Retrieve Call Service or an Answer Call Service.

The Reconnect Call Service request is acknowledged (Ack) by the switch if the switch is able to retrieve the specified held `heldCall` or answer the specified alerting `heldCall`. The request is negatively acknowledged if switch fails to retrieve or answer `heldCall`.

The switch continues to retrieve or answer `heldCall`, even if it fails to drop `activeCall`.³

If the request is negatively acknowledged, the `activeCall` will not be in the active state, if it was in the active state.

3. A race condition may exist between human operation and the application request. The `activeCall` may be dropped before the service request is received by the switch. Since a station can have only one active call, the reconnect operation continues when the switch fails to drop the `activeCall`. If the `activeCall` cannot be dropped because a wrong connection is specified and there is another call active at the station, the retrieve `heldCall` operation will fail.

Service Parameters:

- activeCall*** [mandatory] A valid connection identifier that indicates the callID and the station extension (STATIC_ID). The deviceID in activeCall must contain the station extension of the controlling device. The local connection state of the call must be active.
- heldCall*** [mandatory] A valid connection identifier that indicates the callID and the station extension (STATIC_ID). The deviceID in heldCall must contain the station extension of the controlling device. The local connection state of the call can be either alerting, bridged, or held.

Private Parameters:

dropResource [optional] Specifies the resource to be dropped from the call. The available resources are DR_CALL_CLASSIFIER and DR_TONE_GENERATOR. The tone generator is any G3 PBX applied denial tone that is timed by the switch.

userInfo [optional] Contains user-to-user information. This parameter allows the application to associate caller information, up to 32 or 96 bytes, with a call. It may be a customer number, credit card number, alphanumeric digits, or a binary string.

It is propagated with the call when the call is dropped and passed to the application in a Connection Cleared Event Report. A NULL indicates that this parameter is not present.

Prior to G3V8, the maximum length of userInfo was 32 bytes. Beginning with G3V8, the maximum length of userInfo was increased to 96 bytes.

⇒ NOTE:

An application using private data version 5 and earlier can only receive a maximum of 32-byte data for userInfo, regardless of the size data that is sent by the switch.

The following UUI protocol types are supported:

- UUI_NONE — There is no data provided in the data parameter.
- UUI_USER_SPECIFIC — The content of the data parameter is a binary string. The correct size (maximum of 32 or 96 bytes) of data must be specified in the size parameter.
- UUI_IA5_ASCII — The content of the data parameter must be a null-terminated IA5 (ASCII) character string. The correct size (maximum of 32 or 96 bytes excluding the null terminator) of data must be specified in the size parameter.

Ack Parameter:

noData None for this service.

Nak Parameter:***universalFailure***

If the request is not successful, the application will receive a `CSTAUniversalFailureConfEvent`. The error parameter in this event may contain the following error values, or one of the error values described in the “`CSTAUniversalFailureConfEvent`” section in Chapter 3:

- `GENERIC_UNSPECIFIED` (0) The specified data provided for the `userInfo` parameter exceeds the maximum allowable size. Prior to G3V8, the maximum length of `userInfo` was 32 bytes. Beginning with G3V8, the maximum length of `userInfo` was increased to 96 bytes. See the description of the `userInfo` parameter.
- `INVALID_CSTA_DEVICE_IDENTIFIER` (12) An invalid device identifier or extension is specified in `heldCall`.
- `INVALID_CSTA_CONNECTION_IDENTIFIER` (13) An incorrect `callID` or an incorrect `deviceID` is specified in `heldCall`.
- `GENERIC_STATE_INCOMPATIBILITY` (21) The station user did not go off-hook for `heldCall` within five seconds and is not capable of being forced off-hook.
- `INVALID_CONNECTION_ID_FOR_ACTIVE_CALL` (23) The controlling `deviceIDs` in `activeCall` and `heldCall` are different.
- `INVALID_OBJECT_STATE` (22) The specified `activeCall` at the station is not currently active (in alerting or held state) so it cannot be dropped. The Reconnect Call Service operation stops and the `heldCall` will not be retrieved.

The specified `heldCall` at the station is not in the alerting, connected, held, or bridged state.

- `NO_CALL_TO_ANSWER` (28) The call was redirected to coverage within the five-second interval.
- `GENERIC_SYSTEM_RESOURCE_AVAILABILITY` (31) The switch is busy with another CSTA request. This can happen when two G3PDs are issuing requests (for example, Clear Connection, etc.) to the same device.

The client attempted to add a seventh party to a call with six active parties.

- `RESOURCE_BUSY` (33) User at the station is busy on a call or there are no idle appearances available.
- `MISTYPED_ARGUMENT_REJECTION` (74) `DYNAMIC_ID` is specified in `heldCall`.

Issue 1 — December 2001

Detailed Information:

See the “Detailed Information:” in the “Answer Call Service” section, “Detailed Information:” in the “Clear Connection Service” section and “Detailed Information:” in the “Retrieve Call Service” section in this chapter.

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaReconnectCall() - Service Request

RetCode_t      cstaReconnectCall (
    ACSHandle_t      acsHandle,
    InvokeID_t      invokeID,
    ConnectionID_t  *heldCall,          // devIDType= STATIC_ID
    ConnectionID_t  *activeCall,       // devIDType= STATIC_ID
    PrivateData_t   *privateData);

// CSTAReconnectCallConfEvent - Service Response

typedef struct
{
    ACSHandle_t      acsHandle;
    EventClass_t     eventClass; // CSTACONFIRMATION
    EventType_t     eventType; // CSTA_RECONNECT_CALL_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t  eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
            union
            {
                CSTAReconnectCallConfEvent_t  reconnectCall;
            } u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAReconnectCallConfEvent_t {
    Nulltype      null;
} CSTAReconnectCallConfEvent_t;

```

Private Data Version 6 Syntax

```

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attV6ReconnectCall() - Service Request Private Data
// Setup Function

RetCode_t attV6ReconnectCall(
    ATTPrivateData_t *privateData,
    ATTDropResource_t dropResource); // NULL indicates
// no dropResource
// specified
    ATTUserToUserInfo_t *userInfo); // NULL indicates
// no userInfo
// specified

typedef struct ATTPrivateData_t {
    char vendor[32];
    ushort length;
    char data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef enum ATTDropResource_t {
    DR_NONE = -1, // indicates not specified
    DR_CALL_CLASSIFIER = 0, // call classifier to be dropped
    DR_TONE_GENERATOR = 1 // tone generator to be dropped
} ATTDropResource_t;

#define ATT_MAX_USER_INFO 129
#define ATT_MAX_UII_SIZE 96
#define ATTV5_MAX_UII_SIZE 32

typedef struct ATTUserToUserInfo_t {
    ATTUIIProtocolType_t type;
    struct {
        short length; // 0 indicates UII not present

        unsigned char value[ATT_MAX_USER_INFO];
    } data;
} ATTUserToUserInfo_t;

typedef enum ATTUIIProtocolType_t {
    UII_NONE = -1, // indicates not specified
    UII_USER_SPECIFIC = 0, // user-specific
    UII_IA5_ASCII = 4 // null-terminated ascii
// character string
} ATTUIIProtocolType_t;

```

Private Data Version 2-5 Syntax

```

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attReconnectCall() - Service Request Private Data
// Setup Function

RetCode_t attReconnectCall(
    ATTPrivateData_t *privateData,
    ATTDropResource_t dropResource); // NULL indicates
                                     // no dropResource
                                     // specified
    ATTUserToUserInfo_t *userInfo); // NULL indicates
                                     // no userInfo
                                     // specified

typedef struct ATTPrivateData_t {
    char vendor[32];
    ushort length;
    char data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef enum ATTDropResource_t {
    DR_NONE = -1, // indicates not specified
    DR_CALL_CLASSIFIER = 0, // call classifier to be dropped
    DR_TONE_GENERATOR = 1 // tone generator to be dropped
} ATTDropResource_t;

typedef struct ATTUUIProtocolType_t {
    ATTUUIProtocolType_t type;
    struct {
        short length; // 0 indicates UUI not present
        unsigned char value[33];
    } data;
} ATTUUIProtocolType_t;

typedef enum ATTUUIProtocolType_t {
    UUI_NONE = -1, // indicates not specified
    UUI_USER_SPECIFIC = 0, // user-specific
    UUI_IA5_ASCII = 4 // null-terminated ascii
                       // character string
} ATTUUIProtocolType_t;

```

Retrieve Call Service

Direction: Client to Switch
Function: *cstaRetrieveCall ()*
Confirmation Event: *CSTARRetrieveCallConfEvent*
Service Parameters: *heldCall*
Ack Parameters: *noData*
Nak Parameter: *universalFailure*

Functional Description:

The Retrieve Call Service connects an on-PBX held connection.

Service Parameters:

heldCall [mandatory] A valid connection identifier that indicates the endpoint to be connected. The deviceID in heldCall must contain the station extension of the endpoint.

Ack Parameter:

noData None for this service.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- INVALID_CSTA_DEVICE_IDENTIFIER (12) An invalid device identifier or extension is specified in heldCall.
- INVALID_CSTA_CONNECTION_IDENTIFIER (13) The connectionID contained in the request is invalid.
- GENERIC_STATE_INCOMPATIBILITY (21) The user was on-hook when the request was made and he/she did not go off-hook within five seconds (call remains on hold).
- NO_ACTIVE_CALL (24) The specified call at the station is cleared so it cannot be retrieved.
- NO_HELD_CALL (25) The specified connection at the station is not in the held state (for example, alerting state) so it cannot be retrieved.
- RESOURCE_BUSY (33) The switch is busy with another CSTA request. This can happen when two G3PDs are issuing requests (for example, Hold Call, Retrieve Call, Clear Connection, Conference Call, etc.) to the same device.
- CONFERENCE_MEMBER_LIMIT_EXCEEDED (38) The client attempted to add a seventh party to a six-party conference call.
- MISTYPED_ARGUMENT_REJECTION (74) DYNAMIC_ID is specified in heldCall.

Detailed Information:

- Active State — If the party is already retrieved on the specified call when the switch receives the request, a positive acknowledgment is returned.

- Bridged Call Appearance — Retrieve Call Service is not permitted on parties in the bridged state and may also be more restrictive if the principal of the bridge has an analog station or the exclusion option is in effect from a station associated with the bridge or PCOL.
- Hold State — Normally, the party to be retrieved has been placed on hold from the station or via the Hold Call Service.
- Switch Operation — A party may be retrieved only to the same call from which it had been put on hold as long as there is no other active call at the user's station.

If the user is on-hook (in the held state), the switch must be able to force the station off-hook or the user must go off-hook within five seconds after requesting a Retrieve Call Service. If one of the above conditions is not met, the request is denied (GENERIC_STATE_INCOMPATIBILITY) and the party remains held.

If the user is listening to dial tone while a request for Retrieve Call Service is received, the dial tone will be dropped and the user reconnected to the held call.

If the user is listening to any other kind of tone (for example, denial) or is busy talking on another call, the Retrieve Call Service request is denied (RESOURCE_BUSY).

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaRetrieveCall() - Service Request

RetCode_t      cstaRetrieveCall (
    ACSHandle_t      acsHandle,
    InvokeID_t      invokeID,
    ConnectionID_t  *heldCall,          // devIDType= STATIC_ID
    PrivateData_t   *privateData);

// CSTARetrieveCallConfEvent - Service Response

typedef struct
{
    ACSHandle_t      acsHandle;
    EventClass_t     eventClass; // CSTACONFIRMATION
    EventType_t     eventType; // CSTA_RETRIEVE_CALL_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t  eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
            union
            {
                CSTARetrieveCallConfEvent_t  retrieveCall;
            } u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTARetrieveCallConfEvent_t {
    Nulltype      null;
} CSTARetrieveCallConfEvent_t;

```

Send DTMF Tone Service (Private Data Version 4 and Later)

Direction: Client to Switch

Function: *cstaEscapeService()*

Confirmation Event: *CSTAEscapeServiceConfEvent*

Private Data Function: *attSendDTMFToneExt()* (private data version 5 and later), *attSendDTMFTone()* (private data version 4)

Service Parameters: *noData*

Private Parameters: *sender, receivers, tones, toneDuration, pauseDuration*

Ack Parameters: *noData*

Nak Parameter: *universalFailure*

Functional Description:

The Send DTMF Tone Service on behalf of an on-PBX endpoint sends a sequence of DTMF tones (maximum of 32) to endpoints on the call. The endpoints receiving the DTMF signal can be on-PBX or off-PBX. To send the DTMF tones, the call must be in an established state.

The allowed DTMF tones are digits 0-9 and # and *. Through such a tone sequence, an application could interact with far-end applications, such as automated bank tellers, automated attendants, voice mail systems, database systems, paging services, etc.

A CSTA Confirmation will be returned to the application when the service request has been accepted or when transmission of the DTMF tones has started. No event or indication will be provided to the application when the transmission of the DTMF tones is completed.

Service Parameters:

noData None for this service.

Private Parameters:

sender [mandatory] Specifies the connectionID of the endpoint on whose behalf DTMF tones are to be sent. This connectionID can be an on-PBX endpoint or an off-PBX endpoint (via trunk connection) on the call.

receivers [optional — not supported] A list of up to five connectionIDs that can receive the DTMF tones. If this list is empty (NULL or the count is 0), all parties on the call will receive the DTMF tones if eligible (that is, the voice path allows the party to receive the signals). This parameter is reserved for future use. If present, it will be ignored.

tones [mandatory] DTMF sequence to be generated. The maximum tone sequence that can be sent is 32. The allowed DTMF tones are null-terminated ASCII string with digits 0-9, '#' and '*' only. Any other character in tones is invalid and will cause the request to be denied.

toneDuration [optional] Specifies the number of one hundredth of a second (for example, 10 means 1/10 of a second) used to control the tone duration. The only valid values for the duration are 6 through 35 (one hundredths of a second).

pauseDuration [optional] Specifies the number of one hundredth of a second used to control the pause duration. The only valid values are 4 through 10 (one hundredths of a second)

Ack Parameter:

noData None for this service.

Nak Parameter:

- universalFailure*** If the request is not successful, the application will receive a `CSTAUniversalFailureConfEvent`. The error parameter in this event may contain the following error values, or one of the error values described in the “`CSTAUniversalFailureConfEvent`” section in Chapter 3:
- `VALUE_OUT_OF_RANGE` (3) (CS0/100) The tones parameter has length equal to 0 or greater than 32 or invalid characters are specified in tones. Also, could indicate that parameter values for either *toneDuration* or *pauseDuration* were incorrectly set.
 - `OBJECT_NOT_KNOWN` (4) (CS0/96) Mandatory parameter is missing.
 - `INVALID_CSTA_DEVICE_IDENTIFIER` (13) (CS0/28) Invalid deviceID is specified in sender.
 - `INVALID_OBJECT_STATE` (22) (CS0/98, CS3/63) The service is requested on a call that is currently receiving switch-provided tone, such as dial tone, busy tone, ringback tone, intercept tone, Music-on-Hold/Delay, etc. The call must be in an established state in order to send DTMF tones.
 - `NO_ACTIVE_CALL` (24) (CS3/86) Invalid callID is specified in sender or receivers.
 - `GENERIC_SUBSCRIBED_RESOURCE_AVAILABILITY` (41) (CS0/50) This service is requested on a G3 PBX administered as a release earlier than G3V4.

Detailed Information:

- * And # Characters — If * and/or # characters are present, they will not be interpreted as termination characters or have any other transmission control function.
- AUDIX — AUDIX analog line ports connected to the G3 PBX will be able to receive DTMF tones generated by this service. However, embedded AUDIX or embedded AUDIX configured to emulate an analog line port interface is not supported.
- Call State — This service may be requested for any active call. This service will be denied when this feature is requested on a call that is currently receiving any switch-provided tone, such as busy, ringback, intercept, music-on-hold, etc.
- Connection State — A sender must have an active voice path to the call. A sender at alerting or held local state cannot send the DTMF tone. A receiver must have an active voice path to the sender. A receiver at hold local state will not receive the tone, although the switch will attempt to send the tone.

Issue 1 — December 2001

- DTMF Receiver — Only parties connected to the switch via analog line ports, analog trunk ports (including tie trunks), or digital trunk ports (including ISDN trunk ports) can be a receiver.
- DTMF Sender — Any voice station or (incoming) trunk caller on an active call can be a sender. DTMF tones will be sent to all parties (receivers) with proper connection type except the sender.
- Multiple Send DTMF Tone Requests — An application can send on behalf of different endpoints in a conference call such that DTMF tone sequences overlap or interfere with each other. An application is responsible for ensuring that it does not ask for multiple send DTMF tone requests from multiple parties on the same call at nearly the same time.
- Unsupported DTMF Tones — Tones corresponding to characters A, B, C, D are not supported by this service.
- Tone Cadence and Level — The application can only control the sequence of DTMF tones. The cadence and levels at which the tones are generated will be controlled by the G3 PBX system administration and/or current defaults for the tone receiving ports, rather than by the application. When DTMF tones are sent to a multi-receiver call, the receivers may hear DTMF sequence with differing cadences.
- Service Availability — This service is only available on a G3 PBX with G3V4 or later software.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaEscapeService() - Service Request

RetCode_t      cstaEscapeService (
    ACSHandle_t    acsHandle,
    InvokeID_t     invokeID,
    PrivateData_t  *privateData);

// CSTAEscapeServiceConfEvent - Service Response

typedef struct
{
    ACSHandle_t    acsHandle;
    EventClass_t   eventClass; // CSTACONFIRMATION
    EventType_t    eventType; // CSTA_ESCAPE_SVC_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t    eventHeader;
    union
    {
        struct
        {
            InvokeID_t    invokeID;
            union
            {
                CSTAEscapeSvcConfEvent_t    escapeService;
            } u;
        } cstaConfirmation;
    } event;
} CSTAEvent_t;

typedef struct CSTAEscapeSvcConfEvent_t {
    Nulltype    null;
} CSTAEscapeSvcConfEvent_t;
```

Private Data Version 5 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attSendDTMFToneExt() - Service Request Private Data
// Setup Function

RetCode_t attSendDTMFToneExt(
    ATTPrivateData_t *privateData,
    ConnectionID_t *sender; // mandatory - NULL is
                           // treated as not specified
    ATTCConnIDList_t *receivers; // ignored - reserved for
                                  // future use (send to all
                                  // parties)
    char *tones // mandatory - NULL is
                // treated as not specified
    short toneDuration, // ignored - reserved for
                       // future use
    short pauseDuration); // ignored - reserved for
                           // future use

typedef struct ATTPrivateData_t {
    char vendor[32];
    ushort length;
    char data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef struct ATTCConnIDList_t
{
    int count;
    ConnectionID_t *pParty;
} ATTCConnIDList_t;
```

Private Data Version 4 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attSendDTMFToneExt() - Service Request Private Data
// Setup Function

RetCode_t attSendDTMFToneExt(
    ATTPrivateData_t *privateData,
    ConnectionID_t *sender;           // mandatory - NULL is
                                     // treated as not specified
    ATTV4ConnIDList_t *receivers;    // ignored - reserved for
                                     // future use (send to all
                                     // parties)
    char *tones                       // mandatory - NULL is
                                     // treated as not specified
    short toneDuration,              // ignored - reserved for
                                     // future use
    short pauseDuration);           // ignored - reserved for
                                     // future use

typedef struct ATTPrivateData_t {
    char vendor[32];
    ushort length;
    char data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef struct ATTV4ConnIDList_t
{
    short count;                     // 0 means not specified
                                     // (send to all parties)
    ConnectionID_t party[ATT_MAX_RECEIVERS];
} ATTV4ConnIDList_t;
```


Selective Listening Hold Service (Private Data Version 5 and Later)

Direction: Client to Switch

Function: *cstaEscapeService()*

Confirmation Event: *CSTAEscapeServiceConfEvent*

Private Data Function: *attSelectiveListeningHold()* (private data version 5 and later)

Service Parameters: *noData*

Private Parameters: *subjectConnection, allParties, selectedParty*

Ack Parameters: *noData*

Nak Parameter: *universalFailure*

Functional Description:

The Selective Listening Hold Service allows a client application to prevent a specific party on a call from hearing anything said by another specific party or all other parties on the call. It allows a client application to put a party's (subjectConnection) listening path to a selected party (selectedParty) on listen-hold, or all parties on an active call on listen-hold. The selected party or all parties may be stations or trunks. A party that has been listen-held may continue to talk and be heard by other connected parties on the call since this service does not affect the talking or listening path of any other party. A party will be able to hear parties on the call from which it has not been listen-held, but will not be able to hear any party from which it has been listen-held. This service will also allow the listen-held party to be retrieved (i.e., to again hear the other party or parties on the call).

Service Parameters:

noData None for this service.

Private Parameters:

subjectConnection [mandatory] Specifies the connectionID of the party who will not hear the voice from all other parties or a single party specified in the selectedParty. This connectionID can be an on-PBX endpoint or an off-PBX endpoint (via trunk connection) on the call.

allParties [mandatory] Specifies either all parties' or a single party's listening path is to be held from the subjectConnection party.

- True — the listening paths of all parties on the call will be held from the subjectConnection party. This prevents the subjectConnection from listening to all other parties on the call. The subjectConnection endpoint can still talk and be heard by all other connected parties on the call. The selectedParty parameter is ignored.
- False — the listening path of the subjectConnection party will be held from the selectedParty party. This prevents the subjectConnection from listening to all other parties on the call. The subjectConnection endpoint can still talk and be heard by all other connected parties on the call. The selectedParty parameter must be specified.

selectedParty [optional] A connectionID whose voice will not be heard by the subjectConnection party. If allParties is false, a connectionID must be specified. If allParties is true, the connectionID in this parameter is ignored.

Ack Parameter:

noData None for this service.

Nak Parameter:

- universalFailure*** If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:
- VALUE_OUT_OF_RANGE (3) (CS0/100) A party specified is not part of the call or in wrong state (e.g., a two-party call with the selectedParty still in the alerting state).
 - OBJECT_NOT_KNOWN (4) (CS0/96) Mandatory parameter is missing.
 - INVALID_CSTA_DEVICE_IDENTIFIER (13) (CS0/28) The party specified is not supported by this service (e.g., announcements, extensions without hardware, etc).
 - INVALID_OBJECT_STATE (22) (CS0/98) The request to listen-hold from all parties is not granted because there are no other eligible parties on the call (including any that were previously listen-held).
 - NO_ACTIVE_CALL (24) (CS3/63) Invalid callID is specified.
 - GENERIC_SYSTEM_RESOURCE_AVAILABILITY (31) (CS3/40) Switch capacity has been exceeded.
 - GENERIC_SUBSCRIBED_RESOURCE_AVAILABILITY (41) (CS0/50) This service has not been administratively enabled on the switch.

Detailed Information:

- Announcements — A party cannot be listen-held from an announcement. When a request is made to listen- hold all parties on a call, and there are more parties than just the announcement, the other parties will be listen-held, but the announcement will not. When the only other party on the call is an announcement, the request will fail.
- Attendants —This feature will not work with attendants.
- Call Vectoring — A call cannot be listen-held when in vector processing.
- Conference and Transfer Call — When two calls are conferenced/transferred, the listen-held state of one party (A) from another party (B) in the resulting call is determined as follows:
 1. If party A was listen-held from party B in at least one of the original calls prior to the conference/transfer, party A will remain listen-held from party B in the resulting call.
 2. Otherwise party A will not be listen-held from party B.

When the request is received for a multi-party conference and one of the parties is still alerting, the request will be positively acknowledged and the alerting party will be listen-held upon answering.

- **Converse Agent** — A converse agent may be listen-held. While in this state, the converse agent will hear any DTMF digits that might be sent by the switch (as specified by the switch administration).
- **DTMF Receiver** — When a party has been listen-held while DTMF digits are being transmitted by the same switch (as a result of the Send DTMF service), the listen-held party will still hear the DTMF digits. However, the listen-held party will not hear the DTMF digits if the digits are sent by another switch.
- **Hold Call** — A party that is listen-held may be put on hold and retrieved as usual. A party that is already on hold and is being listen-held will be listen-held after having been retrieved. The service request on a held party will be positively acknowledged.
- **Music On Hold** — Music on Hold ports may not be listen-held (connection is not addressable). If a party is being listen-held from all other parties (while listening to Music on Hold), the party will still hear the Music on Hold.
- **Park/Unpark Call** — A call with parties listen-held may be parked. When the call is unparked, the listening paths that were previously held will remain on listen-hold.
- **Retrieve Call** — If a listen-held party goes on hold and then is retrieved, all listening paths that were listen-held will remain listen-held.
- **Switch Administration** — The Selective Listening Hold Service must be enabled (set to 'y') in order for it to work.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaEscapeService() - Service Request

RetCode_t      cstaEscapeService (
    ACSHandle_t    acsHandle,
    InvokeID_t     invokeID,
    PrivateData_t  *privateData);

// CSTAEscapeServiceConfEvent - Service Response

typedef struct
{
    ACSHandle_t    acsHandle;
    EventClass_t   eventClass; // CSTACONFIRMATION
    EventType_t    eventType; // CSTA_ESCAPE_SERVICE_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t  eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;
```

Private Data Version 5 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attSelectiveListeningHold() - Service Request Private
// Data Setup Function

RetCode_t attSelectiveListeningHold(
    ATTPrivateData_t*privateData,
    ConnectionID_t *subjectConnection,
    Boolean allParties;
    ConnectionID_t *selectedParty);

typedef struct ATTPrivateData_t {
    char vendor[32];
    ushort length;
    char data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

// attSelectiveListeningHoldConfEvent - Private Data
// Service Response

typedef struct ATTSelectiveListeningHoldConfEvent_t {
    Nulltype null;
} ATTSelectiveListeningHoldConfEvent_t;
```

Selective Listening Retrieve Service (Private Data Version 5 and Later)

Direction: Client to Switch

Function: *cstaEscapeService()*

Confirmation Event: *CSTAEscapeServiceConfEvent*

Private Data Function: *attSelectiveListeningRetrieve()*

Service Parameters: *noData*

Private Parameters: *subjectConnection, allParties, selectedParty*

Ack Parameters: *noData*

Nak Parameter: *universalFailure*

Functional Description:

The Selective Listening Retrieve Service allows a client application to retrieve a party (*subjectConnection*) from listen-hold to another party (*selectedParty*) or all parties that were previously being listen-held.

Service Parameters:

noData None for this service.

Private Parameters:

subjectConnection [mandatory] Specifies the connectionID of the party whose listening path will be reconnected to all parties or party specified in the selectedParty. This connectionID can be an on-PBX endpoint or an off-PBX endpoint (via trunk connection) on the call.

allParties [mandatory] Specifies either all parties' or a single party's listening path is to be reconnected from the subjectConnection party.

- True — the listening paths of all parties on the call will be reconnected from the subjectConnection party. This allows the subjectConnection endpoint to be able to listen to all other parties on the call. The selectedParty parameter is ignored.
- False — the listening path of the subjectConnection party will be reconnected from the selectedParty party. This allows the subjectConnection endpoint be able to listen to selectedParty party. The selectedParty parameter must be specified.

selectedParty [optional] A connectionID whose listening path will be retrieved from listen-held by the subjectConnection party. If allParties is false, connectionIDs must be specified. If allParties is true, the connectionID in this parameter is ignored.

Ack Parameter:

noData None for this service.

Nak Parameter:

universalFailure

If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- VALUE_OUT_OF_RANGE (3) (CS0/100) A party specified is not part of the call or in the wrong state (e.g., a two-party call with the selectedParty still in the alerting state).
- OBJECT_NOT_KNOWN (4) (CS0/96) Mandatory parameter is missing.
- INVALID_CSTA_DEVICE_IDENTIFIER (13) (CS0/28) The party specified is not supported by this feature (e.g., announcements, extensions without hardware, etc).
- NO_ACTIVE_CALL (24) (CS3/63) Invalid callID is specified.
- GENERIC_SUBSCRIBED_RESOURCE_AVAILABILITY (41) (CS0/50) This service has not been administratively enabled on the switch.

Detailed Information:

See “Detailed Information:” in the "Selective Listening Hold Service" section in this chapter for details.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaEscapeService() - Service Request

RetCode_t      cstaEscapeService (
    ACSHandle_t    acsHandle,
    InvokeID_t    invokeID,
    PrivateData_t  *privateData);

// CSTAEscapeServiceConfEvent - Service Response

typedef struct
{
    ACSHandle_t    acsHandle;
    EventClass_t   eventClass; // CSTACONFIRMATION
    EventType_t    eventType; // CSTA_ESCAPE_SERVICE_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t  eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;
```

Private Data Version 5 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attSelectiveListeningRetrieve() - Service Request Private
// Data Setup Function

RetCode_t attSelectiveListeningRetrieve(
    ATTPrivateData_t*privateData,
    ConnectionID_t *subjectConnection,
    Boolean allParties;
    ConnectionID_t *selectedParty);

typedef struct ATTPrivateData_t {
    char vendor[32];
    ushort length;
    char data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

// attSelectiveListeningRetrieveConfEvent - Private Data
// Service Response

typedef struct ATTSelectiveListeningRetrieveConfEvent_t {
    Nulltype null;
} ATTSelectiveListeningRetrieveConfEvent_t;
```

Single Step Conference Call Service (Private Data Version 5 and Later)

Direction: Client to Switch

Function: *cstaEscapeService()*

Confirmation Event: *CSTAEscapeServiceConfEvent*

Private Data Function: *attSingleStepConferenceCall()*

Private Data Confirmation Event: *ATTSingleStepConferenceCallConfEvent*

Service Parameters: *noData*

Private Parameters: *activeCall, deviceToBeJoin, participationType, alertDestination*

Ack Parameters: *noData*

Ack Private Parameters: *newCall, connList, ucid*

Nak Parameter: *universalFailure*

Functional Description:

The Single Step Conference Call Service will join a new device into an existing call. This service can be repeated to make n-device conference calls (subject to switching function limits). Currently DEFINITY supports six (6) parties on a call.

⇒ NOTE:

Single Step Conference Call Service is not currently supported by an ISDN BRI station.

Service Parameters:

noData None for this service.

Private Parameters:

activeCall [mandatory] A pointer to a connection identifier in the call to which a new device is to be added. This can be any connection on the call.

deviceToBeJoin [mandatory] A pointer to the device identifier that is to be added to the call. This must be either a physical station extension of any type or an extension administered without hardware (AWOH), but not a group extension.

Physical stations may be connected locally (analog, BRI, DCP, etc.) or remotely as Off-Premises stations. AWOH extensions count towards the maximum parties in a call. Trunks cannot be directly added to a call via this feature. Group extensions (e.g., hunt groups, PCOLs, TEGs, etc.) may not be added.

participationType [optional] Specifies the type of participation the added device has in the resulting call. Possible values are:

- PT_ACTIVE — the added device actively participates in the resulting conferenced call. As a result, the flow direction of the deviceToBeJoin's connection will be Transmit and Receive. Thus the added device can listen and talk.
- PT_SILENT — the added device can listen but cannot actively participate (cannot talk) in the resulting conferenced call. As a result, the flow direction of the deviceToBeJoin's connection will be Receive only. Thus the other parties on the call will be unaware that the added device has joined the call (no display updates). This option is useful for applications that may desire to silently conference in devices (e.g., service observing).

⇒ NOTE:

If a party is Single Step Conferenced in with PT_SILENT, holds the call, and then conferences in another party, the PT_SILENT status of the original party is negated (i. e., the original party would then be heard by all other parties).

alertDestination [optional — partially supported] Specifies whether or not the deviceToBeJoin is to be alerted.

- TRUE — deviceToBeJoin will be alerted (with Delivered event) before it joins the call.

⇒ NOTE:

The “TRUE” option is not supported in the current release. If it is specified, the service request will fail with VALUE_OUT_OF_RANGE.

- FALSE — deviceToBeJoin will connect to the existing call without the device being alerted (no Delivered event). Only the “FALSE” option is supported in the current release.

Ack Parameter:

noData None for this service.

Ack Private Parameters:

newCall [mandatory] A connectionID specifies the callID and the deviceID of the joining device. The callID is the same callID as specified in the service request; that is, the callID of the resulting call is not changed.

connList [optional — supported] Specifies the devices on the resulting *newCall*. This includes a count of the number of devices in the conferenced call and a list of connectionIDs and deviceIDs that define each connection in the call.

- If a device is on-PBX, the extension is specified. The extension consists of station or group extensions. Group extensions are provided when the conference is to a group and the conference completes before the call is answered by one of the group members (TEG, PCOL, hunt group, or VDN extension). It may contain alerting or bridged extensions.
- The static deviceID of a queued endpoint is set to the split extension of the queue.
- [optional — partially supported] Specifies whether or not the *deviceToBeJoin* is to be alerted.
- If a party is off-PBX, then its static device identified or its previously assigned trunk identifier is specified.

ucid [optional — supported] Specifies the Universal Call ID (UCID) of *newCall*. The UCID is a unique call identifier across switches and the network. A valid UCID is a null-terminated ASCII character string. If there is no UCID associated with this call, the *ucid* contains the ATT_NULL_UCID (a 20-character string of all zeros). This parameter is supported by private data version 5 and later only.

Nak Parameter:***universalFailure***

If the request is not successful, the application will receive a `CSTAUniversalFailureConfEvent`. The error parameter in this event may contain the following error values, or one of the error values described in the “`CSTAUniversalFailureConfEvent`” section in Chapter 3:

- `VALUE_OUT_OF_RANGE` (3) (CS0/100) A not supported option is specified or some out-of-range value is specified in a parameter.
- `OBJECT_NOT_KNOWN` (4) (CS0/96) Mandatory parameter is missing.
- `INVALID_CALLED_DEVICE` (6) (CS0/28) The `deviceToBeJoin` is not a valid station or an AWOH extension, or an invalid `callID` is specified
- `INVALID_CALLING_DEVICE` (CS3/27) The `deviceToBeJoin` is on-hook when Single Step Conference is initiated. The `deviceToBeJoin` should be in off-hook/autoanswer condition.
- `PRIVILEGE_VIOLATION_ON_SPECIFIED_DEVICE` (8) (CS3/43) The class of restriction on `deviceToBeJoin` is violated.
- `INVALID_CSTA_DEVICE_IDENTIFIER` (12) (CS0/28) The `deviceToBeJoin` is not a valid identifier.
- `INVALID_FEATURE` (15) (CS3/63) This feature is not supported on the switch. The switch software is prior to Release 6.
- `INVALID_OBJECT_TYPE` (18) (CS0/58) Call has conference restriction due to any of the data-related features (e.g., data restriction, privacy, manual exclusion, etc.).
- `GENERIC_STATE_INCOMPATIBILITY` (21) (CS0/18) The `deviceToBeJoin` cannot be forced off-hook and it did not go off-hook within 5 seconds.
- `INVALID_OBJECT_STATE` (22) (CS0/98) The request is made with option `PT_ACTIVE` while the call is in vector processing.
- `RESOURCE_BUSY` (33) (CS0/17) The `deviceToBeJoin` is busy or not in idle state.
- `CONFERENCE_MEMBER_LIMIT_EXCEEDED` (38) (CS3/42) The maximum allowed number of parties on the call has been reached.

Detailed Information:

- **Bridged Call Appearance** — A principal station with bridged call appearance can be single step conferenced into a call. Stations with bridged call appearance to the principal have the same bridged call appearance behavior, that is, if monitored, the station will receive Established And Conferenced Events when it joins the call. The station will not receive a Delivered Event.
- **Call and Device Monitoring Event Sequences** — A successful SingleStepConferenceCall request will generate an Established Event followed by a Conferenced Event for call monitoring and the monitoring of all devices that are involved in the newCall. The Established Event reports the connection state change of the DeviceToBeJoin and the Conferenced Event reports the result of the SingleStepConferenceCall request. All call-associated information (e.g., original calling and called device, UUI, collected digits, etc.) is reported in the Conferenced Event and Established Event. In both events, the cause value is EC_ACTIVE_MONITOR, if PT_ACTIVE was specified in the SingleStepConferenceCall request and EC_SILENT_MONITOR, if PT_SILENT was specified. The confController and addParty parameters in the Conferenced Event have the same device as specified by DeviceToBeJoin.

The single step conference call event sequences are similar to the two-step conference call event sequences with one exception. Since the added party is alerted in the two-step conference call, a Delivered Event is generated. In a single-step conference call scenario, however; the deviceToBeJoin is added onto the call without alerting. Therefore, no Delivered Event is generated.

- **Call State** — The call into which a station is to be conferenced with Single Step Conference Service may be in any state, except the following situation. If the call is in vector processing and the PT_ACTIVE option is specified in the request, the request will be denied with INVALID_OBJECT_STATE. This will avoid interactions with vector steps such as “collect” when a party joins the call and is able to talk. If the PT_SILENT is specified, the request will be accepted.
- **Dropping Recording Device** — If single-step conference is used to add a recording device into a call, the application has the responsibility of dropping the recording device and/or call when appropriate. The DEFINITY switch cannot distinguish between recording devices and real stations, so if a recording device is left in the call with one other party, the DEFINITY switch will leave that call up forever, until one of those parties drops.
- **Drop Button and Last Added Party** — A party added by Single Step Conference Service will never be considered as “last added party” on the call. Thus, parties added through Single Step Conference Service would not be able to be dropped by using the Drop button.

- **Feature Availability** — The Single Step Conference Service is only available on the DEFINITY switch with Release 6 and later software. Software prior to R6 will deny the service request and return INVALID_FEATURE.
- **Primary Old Call in Conferenced Event** — Since the activeCall and the newCall parameters contain the same callID, there is no meaningful primaryOldCall in the Conferenced Event. The callID in primaryOldCall will have the value 0 and the deviceID will have the value "0" with type DYNAMIC.
- **Remote Agent Trunk to Trunk Conference/Transfer** — In this type application, an incoming call with an external caller is routed to a remote agent. The remote agent wants to transfer the call to another agent (also remote). Upon the agent's transfer request at the desktop, an application may use Single Step Conference Service to join a local device into this trunk-to-trunk call. This local device need not be a physical station; it may be a station AWOH. Having added the local station into the call, the application can hold the call and make a call to the new agent, and then transfer the call. The caller is now connected to the second remote agent, and the local station (physical or AWOH) that was used to accomplish the transfer is no longer on the call.
- **State of Added Station** — A station to be conferenced into a call must be idle. A station is considered idle when it has an idle call appearance for call origination. If a station is off-hook idle when the Single Step Conference Service is received, the station is immediately conferenced in. If a station is on-hook idle and it may be forced off-hook, it will be forced off-hook and immediately conferenced in. If a station is on-hook idle and it may not be forced off-hook, the switch will wait 5 seconds for the user to go off-hook. If the user does not go off-hook within 5 seconds, then a negative acknowledgment with GENERIC_STATE_INCOMPATIBILITY is sent.
- **Security** — As long as it is allowed by switch administration, an application can add a party onto a call with Single Step Conference Call Service without any audible signal or visual display to the existing parties on the call. If security is a concern, proper switch administration must be performed.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaEscapeService() - Service Request

RetCode_t      cstaEscapeService (
    ACSHandle_t    acsHandle,
    InvokeID_t     invokeID,
    PrivateData_t  *privateData);

// CSTAEscapeServiceConfEvent - Service Response

typedef struct
{
    ACSHandle_t    acsHandle;
    EventClass_t   eventClass; // CSTACONFIRMATION
    EventType_t    eventType; // CSTA_ESCAPE_SERVICE_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t    eventHeader;
    union
    {
        struct
        {
            InvokeID_t    invokeID;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;
```

Private Data Version 5 Syntax

```

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attSingleStepConferenceCall() - Service Request Private Data
// Setup Function

RetCode_t attSingleStepConferenceCall(
    ATTPrivateData_t *privateData,
    ConnectionID_t *activeCall, // mandatory
    DeviceID_t *deviceToBeJoin, // mandatory
    ATTParticipationType_t participation,
    Boolean alertDestination);

typedef struct ATTPrivateData_t {
    char vendor[32];
    ushort length;
    char data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

// ATTSingleStepConferenceCallConfEvent - Private Data Service Response

typedef struct
{
    ATTEventType_t eventType;
    // ATT_SINGLE_STEP_CONFERENCE_CALL_CONF

    union
    {
        ATTSingleStepConferenceCallConfEvent_t ssconference;
    }u;
    char heap[ATTPRIV_MAX_HEAP];
} ATTEvent_t;

typedef struct Connection_t {
    ConnectionID_t party;
    DeviceID_t staticDevice; // NULL for not present
} Connection_t;

typedef struct ConnectionList_t {
    int count;
    Connection_t *connection;
} ConnectionList_t;

typedef struct ATTSingleStepConferenceCallConfEvent_t {
    ConnectionID_t newCall;
    ConnectionList_t connList;
    ATTUCID_t ucid;
} ATTSingleStepConferenceCallConfEvent_t;

typedef char ATTUCID_t[64];

```

Transfer Call Service

Direction: Client to Switch

Function: *cstaTransferCall ()*

Confirmation Event: *CSTATransferCallConfEvent*

Private Data Confirmation Event: *ATTTransferCallConfEvent*

Service Parameters: *heldCall, activeCall*

Ack Parameters: *newCall, connList*

Ack Private Parameters: *ucid*

Nak Parameter: *universalFailure*

Functional Description:

This service provides the transfer of an existing held call (*heldCall*) and another active or proceeding call (alerting, queued, held, or connected) (*activeCall*) at a device provided that *heldCall* and *activeCall* are not both in the alerting state at the controlling device. The Transfer Call Service merges two calls with connections at a single common device into one call. Also, both of the connections to the common device become Null and their connectionIDs are released. A connectionID that specifies the resulting new connection for the transferred call is provided.

Service Parameters:

- heldCall*** [mandatory] Must be a valid connection identifier for the call that is on hold at the controlling device and is to be transferred to the activeCall. The deviceID in heldCall must contain the station extension of the controlling device.
- activeCall*** [mandatory] Must be a valid connection identifier of an active or proceeding call at the controlling device to which the heldCall is to be transferred. The deviceID in activeCall must contain the station extension of the controlling device.

Ack Parameters:

- newCall*** [mandatory — partially supported] A connection identifier that specifies the resulting new call identifier for the transferred call.
- connList*** [optional — supported] Specifies the devices on the resulting new call. This includes a count of the number of devices in the new call and a list of up to six connectionIDs and up to six deviceIDs that define each connection in the call.
- If a device is on-PBX, the extension is specified. The extension consists of station or group of extensions. Group extensions are provided when the conference is to a group and the conference completes before the call is answered by one of the group members (TEG, PCOL, hunt group, or VDN extension). It may contain alerting extensions.
 - The static deviceID of a queued endpoint is set to the split extension of the queue.
 - If a party is off-PBX, then its static device identifier or its previously assigned trunk identifier is specified.

Ack Private Parameters:

- ucid*** [optional] Specifies the Universal Call ID (UCID) of newCall. The UCID is a unique call identifier across switches and the network. A valid UCID is a null-terminated ASCII character string. If there is no UCID associated with this call, the ucid contains the ATT_NULL_UCID (a 20-character string of all zeros). This parameter is supported by private data version 5 and later only.

Nak Parameter:

universalFailure

If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- **INVALID_CSTA_DEVICE_IDENTIFIER (12)** An invalid device identifier or extension is specified in heldCall or activeCall.
- **INVALID_CSTA_CONNECTION_IDENTIFIER (13)** The controlling deviceID in activeCall or heldCall has not been specified correctly.
- **GENERIC_STATE_INCOMPATIBILITY (21)** Both calls are alerting. Both calls are being service-observed. An active call is in a vector-processing stage.
- **INVALID_OBJECT_STATE (22)** The connections specified in the request are not in the valid states for the operation to take place. For example, it does not have one active call and one held call as required.
- **INVALID_CONNECTION_ID_FOR_ACTIVE_CALL (23)** The callID in activeCall or heldCall has not been specified correctly.
- **RESOURCE_BUSY (33)** The switch is busy with another CSTA request. This can happen when two G3PDs are issuing requests (for example, Hold Call, Retrieve Call, Clear Connection, Transfer Call, etc.) to the same device.
- **CONFERENCE_MEMBER_LIMIT_EXCEEDED (38)** The request attempted to add a seventh party to an existing six-party conference call.
- **MISTYPED_ARGUMENT_REJECTION (74)** DYNAMIC_ID is specified in heldCall or activeCall.

Detailed Information:

- **Analog Stations** — Transfer Call Service will only be allowed if one call is held and the second is active (talking). Calls on hard-held or alerting cannot be affected by a Transfer Call Service.

An analog station will support Transfer Call Service even if the “switch-hook flash” field on the G3 system administered form is set to “no.” A “no” in this field disables the switch-hook flash function, meaning that a user cannot conference, hold, or transfer a call from his/her phone set, and cannot have the call waiting feature administered on the phone set.

- Bridged Call Appearance — Transfer Call Service is not permitted on parties in the bridged state and may also be more restrictive if the principal of the bridge has an analog station or the exclusion option is in effect from a station associated with the bridge or PCOL.
- Trunk to Trunk Transfer — Existing rules for trunk-to-trunk transfer from a station user will remain unchanged for application monitored calls. In such case, transfer requested via Transfer Call Service will be denied. When this feature is enabled, application monitored calls transferred from trunk to trunk will be allowed, but there will be no further event reports (except for the Network Reached, Established, or Connection Cleared Event Reports sent to the application).

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaTransferCall() - Service Request

RetCode_t      cstaTransferCall (
    ACSHandle_t    acsHandle,
    InvokeID_t     invokeID,
    ConnectionID_t *heldCall,           // devIDType= STATIC_ID
    ConnectionID_t *activeCall,        // devIDType= STATIC_ID
    PrivateData_t  *privateData);

// CSTATransferCallConfEvent - Service Response

typedef struct
{
    ACSHandle_t    acsHandle;
    EventClass_t   eventClass; // CSTACONFIRMATION
    EventType_t    eventType; // CSTA_TRANSFER_CALL_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t    eventHeader;
    union
    {
        struct
        {
            InvokeID_t    invokeID;
            union
            {
                CSTATransferCallConfEvent_t    transferCall;
            } u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct Connection_t {
    ConnectionID_t    party;
    DeviceID_t        staticDevice; // NULL for not present
} Connection_t;

typedef struct ConnectionList_t {
    int                count;
    Connection_t       *connection;
} ConnectionList_t;

typedef struct {
    ConnectionID_t     newCall;
    ConnectionListID_t connList;
} CSTATransferCallConfEvent_t;

```

Private Data Version 5 Syntax

```
// ATTTransferCallConfEvent - Service Response Private
// Data (supported by private data version 5 and later only)

typedef struct
{
    ATTEventType_t    eventType; // ATT_TRANSFER_CALL_CONF
    union
    {
        ATTTransferCallConfEvent_t    transferCall;
    }u;
} ATTEvent_t;

typedef struct ATTTransferCallConfEvent_t
{
    ATTUCID_t    ucid;
} ATTTransferCallConfEvent_t;

typedef char ATTUCID_t[64];
```

Set Feature Service Group

5

Overview

These services allow a client application to set switch-controlled features or values on a G3 device.

Centre Vu Computer Telephony (CVCT) supports the following CSTA Services:

- Set Advice Of Charge Service (Private Data V5 and later)
- Set Agent State Service
- Set Billing Rate Service (Private Data V5 and later)
- Set Do Not Disturb Feature Service
- Set Forwarding Feature Service
- Set Message Waiting Indicator Feature Service

Set Advice of Charge Service (Private Data Version 5 and Later)

Direction: Client to Switch
Function: cstaEscapeService()
Confirmation Event: CSTAEscapeConfEvent
Private Data Function: attSetAdviceOfCharge()
Service Parameters: noData
Private Parameters: featureFlag
Ack Parameters: noData
Ack Private Parameters: noData
Nak Parameter: universalFailure

Functional Description:

DEFINITY ECS Release 5 and later software supports the Charge Advice Event feature. To receive Charge Advice Events, an application must first turn the Charge Advice Event feature on using the Set Advice of Charge Service (Private Data V5).

If the Charge Advice Event feature is turned on, a trunk group monitored by a cstaMonitorDevice, a station monitored by a cstaMonitorDevice, or a call monitored by a cstaMonitorCall or cstaMonitorCallsViaDevice will receive Charge Advice Events. However, this will not occur if the Charge Advice Event is filtered out by the privateFilter in the monitor request and its confirmation event.

This service enables the DEFINITY to support the collection of charging units over ISDN Primary Rate Interfaces. See “Detailed Information:” and the “Charge Advice Event (Private)” section in Chapter 9 for more details of this feature.

Service Parameters:

noData None for this service.

Private Parameters:

featureFlag [mandatory] Specify the flag for turning the feature on or off. A value of TRUE will turn the feature on and a value of FALSE will turn the feature off. If the feature is already turned on, subsequent requests to turn the feature on again will receive positive acknowledgements. If the feature is turned off, subsequent requests to turn the feature off again will receive positive acknowledgements.

Ack Parameters:

noData None for this service.

Ack Private Parameters:

noData None for this service.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the section on "CSTAUniversalFailureConfEvent" in Chapter 3:

- INVALID_FEATURE (15) The Set Advice of Charge Service (Private Data V5) is not supported by the switch.
- VALUE_OUT_OF_RANGE (3) The featureFlag contains an invalid value.

Detailed Information:

- The result of a successful Set Advice of Charge Service (Private Data V5 and later) request applies to an ACS Stream. This means that any program using the same acsHandle will be affected. An application must use the private filter to filter out Advice Of Charge Events, if these events are not useful to the application.
- If this feature is heavily used, it will reduce the maximum Busy Hour Call Completions (BHCC) of the DEFINITY.
- If more than 100 calls are in a call clearing state waiting for charging information, the oldest record will not receive final charge information. In this case a value of 0 and a cause value of EC_NETWORK_CONGESTION will be reported in the Advice of Charge Event.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaEscapeService() - Service Request

RetCode_t  cstaEscapeService (
            ACSHandle_t          acsHandle,
            InvokeID_t          invokeID,
            PrivateData_t*privateData);

// CSTAEscapeServiceConfEvent - Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTACONFIRMATION
    EventType_t eventType; // CSTA_ESCAPE_SERVICE_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            InvokeID_t invokeID;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;
```

Private Parameter Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attSetAdviceOfCharge() - Service Request Private Data Setup Function

RetCode_t   attAdviceOfCharge(
            ATTPrivateData      *privateData,
            Boolean              featureFlag);

typedef struct ATTPrivateData_t {
            char                vendor[32];
            ushort              length;
            char                data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;
```

Set Agent State Service

Direction: Client to Switch

Function: cstaSetAgentState()

Confirmation Event: CSTASetAgentStateConfEvent

**Private Data Function: attV6SetAgentState (private data version 6),
attSetAgentStateExt (private data version 5), attSetAgentState (private data
versions 2-4)**

**Service Parameters: device, agentMode, agentID, agentGroup,
agentPassword**

Private Parameters: workMode, reasonCode, enablePending

Ack Parameters: noData

Ack Private Parameters: isPending

Nak Parameter: universalFailure

Functional Description:

This service allows a client to log an ACD agent into or out of a G3 ACD Split and to specify a change of work mode for a G3 ACD agent.

Service Parameters:

- device*** [mandatory] Specifies the agent extension. This must be a valid on-PBX station extension for an ACD agent.
- agentMode*** [mandatory — partially supported] Specifies log in or log out for an Agent into or out of an ACD split, or a change of work mode for an Agent logged into an ACD split:
- **AM_LOG_IN** — Log in the Agent. This does not imply that the Agent is ready to accept calls. The initial mode for the ACD agent can be set via the workMode private parameter (see the private parameter workMode). If the workMode private parameter is not supplied, the initial work mode for the ACD agent will be set to the G3 specific “Auxiliary-Work Mode”.
 - **AM_LOG_OUT** — Log an Agent out of a specific ACD split. The Agent will be unable to accept additional calls for the ACD split.
 - **AM_NOT_READY** — Change the work mode for an Agent logged into an ACD split to “Not Ready” (equivalent to G3 “Auxiliary-Work Mode”), indicating that the Agent is occupied with some task other than serving a call.
 - **AM_READY** — Change the work mode for Agent logged into an ACD split to “Ready”. The Agent in the Ready state is ready to accept calls or is currently busy with an ACD call. The workMode private parameter may be used to set the ACD agent work mode to the ATT specific “Auto-In-Work Mode” or “Manual-In-Work Mode”. If the workMode private parameter is not supplied, the ACD agent work mode will be set to the ATT specific “Auto-In-Work Mode”.
 - **AM_WORK_NOT_READY** — Change the work mode for an Agent logged into an ACD split to “Work Not Ready” (equivalent to G3 PBX “After-Call-Work Mode”). The Agent in the Work Not Ready state is occupied with the task of serving a call after the call has disconnected, and the Agent is not ready to accept additional calls for the ACD split.
 - **AM_WORK_READY** — A change to “Work Ready” is not currently supported for G3 PBX.

- agentID*** [optional] Specifies the Agent login identifier for the ACD agent. This parameter is mandatory when the `agentMode` parameter is `AM_LOG_IN`; otherwise it is ignored. An `agentID` containing a Logical Agent's login Identifier can be used to log in a Logical Agent (Expert Agent Selection [EAS]) when paired with the `agentPassword`.
- agentGroup*** [mandatory] Specifies the ACD agent split to use to log in, log out, or change the agent work mode to "Not Ready", "Ready" or "Work Not Ready". In an Expert Agent Selection (EAS) environment, the `agentGroup` parameter must contain the skill group extension.
- agentPassword*** [optional — partially supported] Specifies a password that allows an ACD agent to log into an ACD Split. This service parameter is only used if `agentMode` is set to `AM_LOG_IN`; otherwise it is ignored. The `agentPassword` can be used to log in a Logical Agent (with EAS) when included with the Logical Agent's login Identifier, the `agentID`.

Private Parameters:

workMode

[optional] Specifies the work mode for the agent as Auxiliary- Work Mode (WM_AUX_WORK), After-Call-Work Mode (WM_AFT_CALL), Auto-In Mode (WM_AUTO_IN), or Manual- In-Work Mode (WM_MANUAL_IN) based on the agentMode service parameter as follows:

- AM_LOG_IN — The workMode private parameter specifies the initial work mode for the ACD agent. Valid values include “Auxiliary-Work Mode” (Default), “After-Call-Work Mode”, “Auto-In Mode”, or “Manual-In Mode”.
- AM_LOG_OUT — The workMode is ignored.
- AM_NOT_READY — The workMode is ignored.
- AM_READY — The workMode private parameter specifies the work mode for the ACD agent. Valid values include “Auto-In-Work Mode” (Default), or “Manual-In-Work Mode”.
- AM_WORK_NOT_READY — The workMode is ignored.

- AM_WORK_READY — The workMode is ignored.
- reasonCode** [optional] Specifies the reason for change of work mode to WM_AUX_WORK or the logged-out (AM_LOG_OUT) state. Valid reason codes are a single digit 1–9. A value of 0 indicates that the reason code is not available. The meaning of the code (1-9) is defined by the application. This parameter is supported by private data version 5 and later only.
- enablePending** [optional] Specifies whether the requested change can be made pending.
- A value of TRUE will enable the pending feature. If the agent is busy on a call when an attempt is made to change the agentMode to AM_NOT_READY or AM_WORK_NOT_READY, and enablePending is set to TRUE, the change will be made *pending* and will take effect as soon as the agent clears the call. The request will be acknowledged (Ack).
 - If enablePending is not set to TRUE and the agent is busy on a call, the requested change will not be made *pending* and the request will not be acknowledged (Nak).

⇒ NOTE:

Subsequent requests may override a pending change and only the most recent pending change will take effect when the call is cleared. The enablePending parameter applies to the reasonCode when the request is to change the agentMode to AM_NOT_READY.

This parameter is supported by private data version 6 and later only.

Ack Parameters:

noData None for this service.

Ack Private Parameters:

isPending [optional]If *isPending* is set to TRUE, the requested change in workmode is pending. Otherwise, the requested change took effect immediately.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a *CSTAUniversalFailureConfEvent*. The error parameter in this event may contain the following error values, or one of the error values described in the section on “*CSTAUniversalFailureConfEvent*” in Chapter 3:

- **GENERIC_UNSPECIFIED (0)** An attempt to log out an ACD agent who is already logged out, an attempt to log in an ACD agent to a split of which they are not a member, or an attempt to log in an ACD agent with an incorrect password.
- **GENERIC_OPERATION (1)** An attempt to log in an ACD agent that is already logged in.
- **VALUE_OUT_OF_RANGE (3)** The *workMode* private parameter is not valid for the *agentMode* (see Table 5-1).

The reason code is out of the acceptable range (1- 9). (CS0/100).
- **OBJECT_NOT_KNOWN (4)** Did not specify a valid on-PBX station for the ACD agent in device, the *agentGroup* or device parameters were NULL, or the *agentID* parameter was NULL when *agentMode* was set to *AM_LOG_IN*.
- **INVALID_CSTA_DEVICE_IDENTIFIER (12)** An invalid device identifier has been specified in device.
- **INVALID_FEATURE (15)** The feature is not available for the *agentGroup*, or the *enablePending* feature is not available for the administered switch version.
- **INVALID_OBJECT_TYPE (18)** (CS3/80) The reason code is specified, but the *workMode* is not *WM_AUX_WORK*, or *agentMode* is not *AM_LOG_OUT*.
- **GENERIC_STATE_INCOMPATIBILITY (21)** A work mode change was requested for a non-ACD agent, or the Agent station is maintenance busy or out of service.

Issue 1 — December 2001

- INVALID_OBJECT_STATE (22) The Agent is already logged into another split.
- GENERIC_SYSTEM_RESOURCE_AVAILABILITY (31) The request cannot be completed due to lack of available switch resources.
- RESOURCE_BUSY (33) Attempt to log in an ACD agent that is currently on a call.

Detailed Information:

- A request to log in an ACD agent (agentMode is AM_LOG_IN) that does not include the private parameter workMode, will set the initial Agent work state to Auxiliary-Work Mode (Not Ready).
- The AM_WORK_READY agentMode is not supported by G3 PBX.
- The agentPassword service parameter applies only for requests to log in an ACD agent (agentMode is AM_LOG_IN). In all other cases, it is ignored. The agentPassword can be used to log in a Logical Agent (with Expert Agent Selection [EAS]) when included with the Logical Agent's login Identifier, the agentID.
- Valid combinations of the agentMode service parameter and the workMode, reasonCode, and enablePending private parameters are shown in Table 5-1.

Table 5-1. AgentMode Service Parameter and Associated Private Parameters

agentMode	workMode	reasonMode	enablePending
AM_LOG_IN	WM_AUX_WORK (Default) WM_AFTCAL_WK WM_AUTO_IN WM_MANUAL_IN	1-9	NA
AM_LOG_OUT	NA	NA	NA
AM_NOT_READY	NA	1-9	TRUE/FALSE
AM_READY	WM_AUTO_IN (Default) WM_MANUAL_IN	NA	NA
AM_WORK_NOT_READY	NA	NA	TRUE/FALSE
AM_WORK_READY	NA	NA	NA

- AttSetAgentStateExt() and attSetAgentState() do not accept the enablePending parameter. These functions will never cause the requested work mode change to be made pending, even if the switch is G3V8 or later.
- Subsequent pending work mode requests replace earlier requests.

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaSetAgentState() - Service Request

RetCode_t  cstaSetAgentState (
    ACSHandle_t  acsHandle,
    InvokeID_t   invokeID,
    DeviceID_t   *device,
    AgentMode_t  agentMode,
    AgentID_t    *agentID,
    AgentGroup_t *agentGroup,
    AgentPassword_t*agentPassword,
    PrivateData_t *privateData);

typedef char          DeviceID_t[64];

typedef enum AgentMode_t {
    AM_LOG_IN           = 0,
    AM_LOG_OUT          = 1,
    AM_NOT_READY        = 2,
    AM_READY             = 3,
    AM_WORK_NOT_READY   = 4,
    AM_WORK_READY        = 5
} AgentMode_t;

typedef char          AgentID_t[32];

typedef DeviceID_t    AgentGroup_t;

typedef char          AgentPassword_t[32];

typedef struct PrivateData_t {
    char                vendor[32];
    unsigned short      length;
    char                data[1]; // actual length determined by
                                // application
} PrivateData_t;

// CSTASetAgentStateConfEvent - Service Response

typedef struct
{
    ACSHandle_t          acsHandle;
    EventClass_t         eventClass; // CSTA_CONFIRMATION
    EventType_t          eventType; // CSTA_SET_AGENT_STATE_CONF
} ACSEventHeader_t;

typedef struct CSTASetAgentStateConfEvent_t {
    Nulltype             null;

```

Syntax (Continued)

```
} CSTASetAgentStateConfEvent_t;

typedef struct
{
    ACSEventHeader_t eventHeader;

    union
    {
        struct
        {
            InvokeID_t invokeID;
            union
            {
                CSTASetAgentStateConfEvent_t setAgentState;
            } u;
        } cstaConfirmation;
        } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;
```


Private Data Version 6 Syntax

```

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attV6SetAgentState() - Service Request Private Data Setup Function

RetCode_t    attV6SetAgentState(
    ATTPrivateData_t*attPrivateData,
    ATTWorkMode_t workMode,           // Work Modes
    long      reasonCode,           // single digit 1-9
    Boolean   enablePending);       // TRUE = enabled

typedef struct ATTPrivateData_t
{
    char      vendor[32];
    ushort   length;
    char      data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef enum ATTWorkMode_t
{
    WM_AUX_WORK    = 1,           // Same As C_AUX_WORK
    WM_AFTCAL_WK   = 2,           // Same as C_AFTCAL_WK
    WM_AUTO_IN     = 3,           // Same as C_AUTO_IN
    WM_MANUAL_IN   = 4           // Same as C_MANUAL_IN
} ATTWorkMode_t;

// ATTSetAgentStateConfEvent - Confirmation Event Private Data

typedef struct
{
    ATTEventType eventType;       // ATT_SET_AGENT_STATE_CONF
    union
    {
        ATTSetAgentStateConfEvent_t setAgentState;
    };
    char      heap[ATTPRIV_MAX_HEAP];
} ATTEvent_t;

typedef struct ATTSetAgentStateConfEvent_t {
    Boolean   isPending; // TRUE if the request is pending
} ATTSetAgentStateConfEvent_t;

```

Private Data Version 5 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attSetAgentState() - Service Request Private Data Setup Function

RetCode_t    attSetAgentState(
    ATTPrivateData_t*attPrivateData,
    ATTWorkMode_t workMode,           // Work Modes
    long       reasonCode);         // single digit 1-9

typedef struct ATTPrivateData_t
{
    char        vendor[32];
    ushort     length;
    char        data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef enum ATTWorkMode_t
{
    WM_AUX_WORK    = 1,           // Same As C_AUX_WORK
    WM_AFTCAL_WK   = 2,           // Same as C_AFTCAL_WK
    WM_AUTO_IN     = 3,           // Same as C_AUTO_IN
    WM_MANUAL_IN   = 4           // Same as C_MANUAL_IN
} ATTWorkMode_t;
```

Private Data Versions 2-4 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attSetAgentState() - Service Request Private Data Setup Function

RetCode_t    attSetAgentState(
              ATTPrivateData_t*attPrivateData,
              ATTWorkMode_t workMode);           // Work Modes

typedef struct ATTPrivateData_t
{
    char        vendor[32];
    ushort     length;
    char        data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef enum ATTWorkMode_t
{
    WM_AUX_WORK    = 1,           // Same As C_AUX_WORK
    WM_AFTCAL_WK   = 2,           // Same as C_AFTCAL_WK
    WM_AUTO_IN     = 3,           // Same as C_AUTO_IN
    WM_MANUAL_IN   = 4           // Same as C_MANUAL_IN
} ATTWorkMode_t;
```

Set Billing Rate Service (Private Data Version 5 and Later)

Direction: Client to Switch
Function: cstaEscapeService()
Confirmation Event: CSTAEscapeConfEvent
Private Data Function: attSetBillingRate()
Service Parameters: noData
Private Parameters: call, billType, billRate
Ack Parameters: noData
Nak Parameter: universalFailure

Functional Description:

This service supports the AT&T MultiQuest[®] 900 Vari-A-Bill Service to change the rate for an incoming 900-type call. The client application can request this service at any time after the call has been answered and before the call is cleared.

Service Parameters:

noData None for this service.

Private Parameters:

call [mandatory] Specifies the call to which the billing rate is to be applied. This is a connection identifier, but only the callID is used. The deviceID of call is ignored.

billType [mandatory] Specifies the rate treatment for the call and can be one of the following:

- BT_NEW_RATE
- BT_FLAT_RATE (i.e., time independent)
- BT_PREMIUM_CHARGE (i.e., a flat charge in addition to the existing rate)
- BT_PREMIUM_CREDIT (i.e., a flat negative charge in addition to the existing rate)
- BT_FREE_CALL

billRate [mandatory] Specifies the rate according to the treatment indicated by billType. If FREE_CALL is specified, billRate is ignored. This is a floating point number. The rate should not be less than \$0 and a maximum is set for each 900-number as part of the provisioning process (in the 4E switch)

Ack Parameters:

noData None for this service.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- INVALID_CSTA_DEVICE_IDENTIFIER (12) An invalid device identifier has been specified in device.
- INVALID_CSTA_CONNECTION_IDENTIFIER (13) An invalid connection identifier has been specified in call.
- VALUE_OUT_OF_RANGE (3) (CS0/96) Invalid value is specified in the request.

- INVALID_OBJECT_STATE (22) (CS0/98) The request is attempted before the call is answered.
- RESOURCE_BUSY (33) (CS0/47) The switch limit for unconfirmed requests has been reached.
- GENERIC_SUBSCRIBED_RESOURCE_AVAILABILITY (41) (CS0/29) The user has not subscribed to the Set Billing Rate Service (Private Data V5 and later) feature.

Detailed Information:

None

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaEscapeService() - Service Request

RetCode_t  cstaEscapeService (
            ACSHandle_t          acsHandle,
            InvokeID_t          invokeID,
            PrivateData_t       *privateData);

// CSTAEscapeServiceConfEvent - Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTACONFIRMATION
    EventType_t eventType;  // CSTA_ESCAPE_SERVICE_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            InvokeID_t invokeID;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;
```

Private Parameter Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attSetBillingRate() - Service Request Private Data Setup Function

RetCode_t  attSetBillingRate(
            ATTPrivateData      *privateData,
            ConnectionID_t      *call,
            ATTBillType_t       billType,
            float                billRate);

typedef struct ATTPrivateData_t {
    char        vendor[32];
    ushort     length;
    char        data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef enum ATTBillType_t {
    BT_NEW_RATE           = 16,
    BT_FLAT_RATE          = 17,
    BT_PREMIUM_CHARGE     = 18,
    BT_PREMIUM_CREDIT     = 19,
    BT_FREE_CALL          = 24
} ATTBillType_t;
```


Set Do Not Disturb Feature Service

Direction: Client to Switch

Function: `cstaSetDoNotDisturb()`

Confirmation Event: `CSTASetDndConfEvent`

Service Parameters: `device`, `doNotDisturb`

Ack Parameters: `noData`

Nak Parameter: `universalFailure`

Functional Description:

This service turns on or off the G3 Send All Calls (SAC) feature for a user station.

Service Parameters:

device [mandatory] Must be a valid on-PBX station extension that supports the SAC feature.

doNotDisturb [mandatory] Specifies either “On” (TRUE) or “Off” (FALSE).

Ack Parameters:

noData None for this service.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a `CSTAUniversalFailureConfEvent`. The error parameter in this event may contain the following error values, or one of the error values described in the “`CSTAUniversalFailureConfEvent`” section in Chapter 3:

- `INVALID_CSTA_DEVICE_IDENTIFIER` (12) An invalid device identifier has been specified in device.
- `GENERIC_SUBSCRIBED_RESOURCE_AVAILABILITY` (41) The user has not subscribed to the SAC feature.

Detailed Information:

- **DCS** — SAC feature may not be requested by this service for an off-PBX DCS extension.
- **Logical Agents** — SAC may not be requested by this service for logical agent login IDs. If a login ID is specified, the request will be denied (`INVALID_CSTA_DEVICE_IDENTIFIER`). SAC may be requested by this service on behalf of a logical agent’s station extension.

In an Expert Agent Selection (EAS) environment, if the call is made to a logical agent ID, the call coverage follows the path administered for the logical agent ID, and not the coverage path of the physical set from which the agent is logged in. SAC cannot be activated by a CSTA request for the logical agent ID.

- **Send All Calls (SAC)** — This G3 feature allows users to temporarily direct all incoming calls to coverage regardless of the assigned Call Coverage redirection criteria. Send All Calls also allows covering users to temporarily remove their voice terminals from the coverage path. SAC is used only in conjunction with the Call Coverage feature. Details of how SAC is used in conjunction with the Call Coverage are documented in the DEFINITY Communications System Generic 3 Feature Description, 555-230-201.

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaSetDoNotDisturb() - Service Request

RetCode_t  cstaSetDoNotDisturb (
    ACSHandle_t acsHandle,
    InvokeID_t  invokeID,
    DeviceID_t  *device,
    Boolean     doNotDisturb, // TRUE = On   or FALSE = Off
    PrivateData_t*privateData);

typedef char      DeviceID_t[64];
typedef char      Boolean;

// CSTASetDndConfEvent - Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass;           // CSTACONFIRMATION
    EventType_t eventType;           // CSTA_SET_DND_CONF
} ACSEventHeader_t;

typedef struct CSTASetDndConfEvent_t {
    Nulltype      null;
} CSTASetDndConfEvent_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
            union
            {
                CSTASetDndConfEvent_t setDnd;
            } u;
            cstaConfirmation;
        } event;
        char      heap[CSTA_MAX_HEAP];
    }
} CSTAEvent_t;

```

Set Forwarding Feature Service

Direction: Client to Switch

Function: cstaSetForwarding()

Confirmation Event: CSTASetFwdConfEvent

Service Parameters: device, forwardingType, forwardingOn, forwardingDN

Ack Parameters: noData

Nak Parameter: universalFailure

Functional Description:

The Set Forwarding Service sets the G3 Call Forwarding feature on or off for a user station. G3 CSTA supports the Immediate type of forwarding only.

Service Parameters:

<i>device</i>	[mandatory] Specifies the station on which the Call Forwarding feature is to be set. It must be a valid on-PBX station extension that supports the Call Forwarding feature.
<i>forwardingType</i>	[mandatory — partial] Specifies the type of forwarding to set or clear. Only FWD_IMMEDIATE is supported. Any other types will be denied.
<i>forwardingOn</i>	[mandatory] Specifies "On" (TRUE) or "Off" (FALSE).
<i>forwardingDN</i>	[mandatory] Specifies the station extension to which the calls are to be forwarded. It is mandatory if forwardingOn is set to on. It is ignored by the G3 switch if the forwardingOn is set to off.

Ack Parameters:

noData None for this service.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- INVALID_CSTA_DEVICE_IDENTIFIER (12) An invalid device identifier has been specified in device or forwardingDN.
- GENERIC_SUBSCRIBED_RESOURCE_AVAILABILITY (41) The user has not subscribed to the Call Forwarding feature.

Detailed Information:

- DCS — The Call Forwarding feature may not be activated by this service for an off-PBX DCS extension.
- Logical Agents — Call Forwarding may not be requested by this service for logical agent login IDs. If a login ID is specified as the forwardingDN, the request will be denied (INVALID_CSTA_DEVICE_IDENTIFIER). Call Forwarding may be requested on behalf of a logical agent's station extension.
- G3 Call Forwarding All Calls — This feature allows all calls to an extension number to be forwarded to a selected internal extension number, external (off-premises) number, the attendant group, or a specific attendant. It supports only the CSTA forwarding type “Immediate.”

- Activation and Deactivation — Activation and deactivation from the station and a client application may be intermixed.

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaSetForwarding() - Service Request

RetCode_t  cstaSetForwarding (
    ACSHandle_t  acsHandle,
    InvokeID_t  invokeID,
    DeviceID_t  *device,
    ForwardingType_t forwardingType, // must be FWD_IMMEDIATE
    Boolean      forwardingOn, // TRUE (on) or FALSE (off)
    DeviceID_t  *forwardingDestination,
    PrivateData_t *privateData);

typedef char      DeviceID_t[64];

typedef enum ForwardingType_t {
    FWD_IMMEDIATE      = 0,          // Only option supported
    FWD_BUSY            = 1,          // Not supported
    FWD_NO_ANS          = 2,          // Not supported
    FWD_BUSY_INT       = 3,          // Not supported
    FWD_BUSY_EXT       = 4,          // Not supported
    FWD_NO_ANS_INT     = 5,          // Not supported
    FWD_NO_ANS_EXT     = 6,          // Not supported
} ForwardingType_t;

typedef char      Boolean;

// CSTASetFwdConfEvent - Service Response

typedef struct
{
    ACSHandle_t      acsHandle;
    EventClass_t     eventClass; // CSTACONFIRMATION
    EventType_t     eventType;  // CSTA_SET_FWD_CONF
} ACSEventHeader_t;

typedef struct CSTASetFwdConfEvent_t {
    Nulltype        null;
} CSTASetFwdConfEvent_t;

```

Syntax (Continued)

```
typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            InvokeID_t invokeID;
            union
            {
                CSTASetFwdConfEvent_t setFwd;
            } u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;
```


Set Message Waiting Indicator (MWI) Feature Service

Direction: Client to Switch
Function: `cstaSetMsgWaitingInd()`
Confirmation Event: `CSTASetMwiConfEvent`
Service Parameters: `device`, `messages`
Ack Parameters: `noData`
Nak Parameter: `universalFailure`

Functional Description:

This service sets the G3 Message Waiting Indicator (MWI) on or off for a user station.

Service Parameters:

device [mandatory] Must be a valid on-PBX station extension that supports the MWI feature.

messages [mandatory] Specifies either “On” (TRUE) or “Off” (FALSE).

Ack Parameter:

noData None for this service.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a `CSTAUniversalFailureConfEvent`. The error parameter in this event may contain the following error value, or one of the error values described in the “`CSTAUniversalFailureConfEvent`” section in Chapter 3:

- `INVALID_CSTA_DEVICE_IDENTIFIER` (12) An invalid device identifier has been specified in `device`.

Detailed Information:

- Adjunct Messages — When a client application has turned on a station’s MWI and the station user retrieves message using the station display, then the station display will show “You have adjunct messages.”
- MWI Status Sync — To keep the MWI synchronized with other applications, a client application must use this service to update the MWI whenever the link between the switch and the PBX Driver comes up from a cold start. An application can query the MWI status through the `CSTAQueryMsgWaitingInd` Service.

- System Starts — System cold starts will cause the switch to lose the MWI status. Hot starts (PE interchange) and warm starts will not affect the MWI status.
- Voice (Synthesized) Message Retrieval — A recording, “Please call message center for more messages,” will be used for the case when the MWI has been activated by the application through this service.

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaSetMsgWaitingInd() - Service Request

RetCode_t  cstaSetMsgWaitingInd (
    ACSHandle_t acsHandle,
    InvokeID_t  invokeID,
    DeviceID_t  *device,
    Boolean     messages,    // TRUE (on) or FALSE (off)
    PrivateData_t*privateData);

typedef char      DeviceID_t[64];
typedef char      Boolean;

// CSTASetMwiConfEvent - Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass;           // CSTACONFIRMATION
    EventType_t eventType;           // CSTA_SET_MWI_CONF
} ACSEventHeader_t;

typedef struct CSTASetMwiConfEvent_t {
    Nulltype     null;
} CSTASetMwiConfEvent_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
            union
            {
                CSTASetMwiConfEvent_t setMwi;
            } u;
            cstaConfirmation;
        } event;
        char      heap[CSTA_MAX_HEAP];
    }
} CSTAEvent_t;

```


Overview

These services allow a client application to query the switch to provide the state of device features and static attributes of a device.

The following CSTA Services are supported:

- Query ACD Split Service
- Query Agent Login Service
- Query Agent State Service
- Query Call Classifier Service
- Query Device Info
- Query Device Name Service
- Query Do Not Disturb Service
- Query Forwarding Service
- Query Message Waiting Service
- Query Station Status Service
- Query Time of Day Service
- Query Trunk Group Service
- Query Universal Call ID

Query ACD Split Service

Direction: Client to Switch Function: cstaEscapeService() Confirmation

Event: CSTAEscapeServiceConfEvent

Private Data Function: attQueryACDSplit()

Private Data Confirmation Event: ATTQueryACDSplitConfEvent

Service Parameters: noData

Private Parameters: device

Ack Parameters: noData

Ack Private Parameters: availableAgents, callsInQueue, agentsLoggedIn

Nak Parameter: universalFailure

Functional Description:

The Query ACD Split service provides the number of ACD agents available to receive calls through the split, the number of calls in queue, and the number of agents logged in. The number of calls in queue does not include direct-agent calls.

Service Parameters:

noData None for this service.

Private Parameters:

device [mandatory] Must be a valid ACD split extension.

Ack Parameters:

noData None for this service.

Ack Private Parameters:

availableAgents [mandatory] Specifies the number of ACD agents available to receive calls through the specified split.

callsInQueue [mandatory] Specifies the number of calls in queue (not including direct-agent calls).

agentsLoggedIn [mandatory] Specifies the number of ACD agents logged in.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- INVALID_CSTA_DEVICE_IDENTIFIER (12) An invalid device identifier has been specified in device.

Detailed Information:

None

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaEscapeService() - Service Request

RetCode_t  cstaEscapeService (
    ACSHandle_tacsHandle,
    InvokeID_tinvokeID,
    PrivateData_t*privateData);

// CSTAEscapeServiceConfEvent - Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_teventClass;// CSTACONFIRMATION
    EventType_teventType;// CSTA_ESCAPE_SERVICE_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_teventHeader;
    union
    {
        struct
        {
            InvokeID_tinvokeID;
            union
            {
                CSTAEscapeSvcConfEventescapeService;
            }u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAEscapeSvcConfEvent_t {
    Nulltypenull
} CSTAEscapeSvcConfEvent_t;
```


Private Parameter Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attQueryACDSplit() - Service Request Private Data Setup
Function

RetCode_t*attQueryACDSplit (// returns NULL if no
// parameter specified
    ATTPrivateData_t*privateData,
    DeviceID_t*device);

typedef struct ATTPrivateData_t {
    char        vendor[32];
    unsigned shortlength;
    char        data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

// ATTQueryACDSplitConfEvent - Service Response Private Data

typedef struct
{
    ATTEventTypeeventType;// ATT_QUERY_ACD_SPLIT_CONF
    union
    {
        ATTQueryACDSplitConfEvent_tqueryACDSplit;
    }u;
    char    heap[ATTPRIV_MAX_HEAP];
} ATTEvent_t;

typedef struct ATTQueryACDSplitConfEvent_t
{
    shortavailableAgents;// number of available agents
// to receive call
    shortcallsInQueue;// number of calls in queue
    shortagentsLoggedIn;// number of agents logged in
} ATTQueryACDSplitConfEvent_t;
```

Query Agent Login Service

Direction: Client to Switch Function: `cstaEscapeService()`
Confirmation Event: `CSTAPrivateEvent`, `CSTAEscapeServiceConfEvent`
Private Data Function: `attQueryAgentLogin()`, `ATTQueryAgentLoginResp()`
Private Data Confirmation Event: `ATTQueryAgentLoginConfEvent`
Service Parameters: `noData`
Private Parameters: `device`
Ack Parameters: `noData`
Ack Private Parameters: `privEventCrossRefID`
Private Event Parameters: `privEventCrossRefID`, `list`
Nak Parameter: `universalFailure`

Functional Description:

The Query Agent Login Service provides the extension of each ACD agent logged into the specified ACD split. This service is unlike most other services because the confirmation event provides a unique private event cross reference ID that associates a subsequent `CSTAPrivateEvent` (containing the actual ACD agent login data) with the original request. The private event cross reference ID is the only data returned in the confirmation event. After returning the confirmation event, the service returns a sequence of `CSTAPrivateEvents`. Each `CSTAPrivateEvent` contains the private event cross reference ID, and a list. The list contains the number of extensions in the message, and up to 10 extensions of ACD agents logged into the ACD split.

The entire sequence of `CSTAPrivateEvents` may contain a large volume of information (up to the maximum number of logged-in agents allowed in an ACD Split). The service provides the private event cross reference ID in case an application has issued multiple Query Agent Login requests. The final `CSTAPrivateEvent` specifies that it contains zero extensions and serves to inform the application that no more messages will be sent in response to this query.

Service Parameters:

noData None for this service.

Private Parameters:

device [mandatory] Must be a valid ACD split extension.

Ack Parameters:

noData None for this service.

Ack Private Parameters:

privEventCrossRefID Contains a unique handle to identify subsequent CSTAPrivateEvents with this request.

Private Event Parameters:

privEventCrossRefID [mandatory] The handle to the query agent login request for which this CSTAPrivateEvent is reported.

list [mandatory] A list structure with the following information: the count (0 - 10) of how many extensions are in the message and an array of up to 10 extensions. A count value of 0 is like an "end of file" - i.e., there are no additional CSTAPrivateEvents for the query.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error value, or one of the error values described in the "CSTAUniversalFailureConfEvent" section in Chapter 3:

- INVALID_CSTA_DEVICE_IDENTIFIER (12) An invalid device identifier has been specified in device.

Detailed Information:

- A single Query Agent Login Request may result in multiple CSTAPrivateEvents returned to the client after the return of the confirmation event. All messages are contained in private data of the CSTAPrivateEvents.
- This service uses a private event cross reference ID to provide a way for clients to correlate incoming CSTAPrivateEvents with an original Query Agent Login request.
- Each separate CSTAPrivateEvent may contain up to 10 extensions.

- Each separate CSTAPrivateEvent contains a number indicating how many extensions are in the message. The last CSTAPrivateEvent has the number set to zero.
- The service receives each response message from the switch and passes it to the application in a CSTAPrivateEvent. The application must be prepared to receive and deal with a potentially large number of extensions received in multiple CSTAPrivateEvents after it receives the confirmation event.

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaEscapeService() - Service Request

RetCode_t  cstaEscapeService (
    ACSHandle_tacsHandle,
    InvokeID_tinvokeID,
    PrivateData_t*privateData);

// CSTAEscapeServiceConfEvent - Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_teventClass;// CSTACONFIRMATION
    EventType_teventType;// CSTA_ESCAPE_SERVICE_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_teventHeader;
    union
    {
        struct
        {
            InvokeID_tinvokeID;
            union
            {
                CSTAEscapeSvcConfEvent_tescapeService;
            }u;
        } cstaConfirmationEvent;
    } event;
    charheap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAEscapeSvcConfEvent_t {
    Nulltypenull
} CSTAEscapeSvcConfEvent_t;

```

Syntax (Continued)

```
// CSTAPrivateEvent - Private event for reporting data

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            union
            {
                CSTAPrivateEvent_t privateData;
            }u;
        } cstaEventReport;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAPrivateEvent_t {
    Null_t penull
} CSTAPrivateEvent_t;
```

Private Parameter Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attQueryAgentLogin() - Service Request Private Data
Setup Function

RetCode_tattQueryAgentLogin ( // returns NULL if no
// parameter specified
    ATTPrivateData_t*privateData,
    DeviceID_t*device);

typedef struct ATTPrivateData_t {
    char        vendor[32];
    unsigned shortlength;
    char        data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

// ATTQueryAgentLoginConfEvent - Confirmation Event
Private Data

typedef struct
{
    ATTEventTypeeventType;// ATT_QUERY_AGENT_LOGIN_CONF
    union
    {
        ATTQueryAgentLoginConfEvent_tqueryAgentLogin;
    }u;
    char heap[ATTPRIV_MAX_HEAP];
} ATTEvent_t;

typedef struct ATTQueryAgentLoginConfEvent_t {
    ATTPrivEventCrossRefID_tprivEventCrossRefID;
} ATTQueryAgentLoginConfEvent_t;

// ATTQueryAgentLoginEvent - Private Event Private Data
typedef struct
{
    ATTEventTypeeventType;// ATT_QUERY_AGENT_LOGIN_RESP
    union
    {
        ATTQueryAgentLoginResp_tqueryAgentLoginResp;
    }u;
    char heap[ATTPRIV_MAX_HEAP];
} ATTEvent_t;
```

Private Parameter Syntax (Continued)

```
typedef struct ATTQueryAgentLoginResp_t
{
    ATTPrivEventCrossRefID_tprivEventCrossRefID;
    // cross reference ID
    struct {
        shortcount;// number of extensions in
        // device[]
        DeviceID_tdevice[10];// up to 10 extensions
    } list;
} ATTQueryAgentLoginResp_t;
```


Query Agent State Service

Direction: Client to Switch Function: cstaQueryAgentState()
Confirmation Event: CSTAQueryAgentStateConfEvent
Private Data Function: attQueryAgentState()
Private Data Confirmation Event: ATTQueryAgentStateConfEvent (private data version 6), ATTV5QueryAgentStateConfEvent (private data version 5), ATTV4QueryAgentStateConfEvent (private data versions 2-4)
Service Parameters: device
Private Parameters: split
Ack Parameters: agentState
Ack Private Parameters: workMode, talkState, reasonCode, pendingWorkMode, pendingReasonCode
Nak Parameter: universalFailure

Functional Description:

This service provides the agent state of an ACD agent. The agent's state is returned in the CSTA AgentState parameter. The private talkState parameter indicates if the agent is idle or busy. The private workMode parameter has the agent's work mode as defined by the DEFINITY PBX. The private reasonCode has the agent's reasonCode if one is set. The private pendingWorkMode and pendingReasonCode have the work mode and reason code that will take effect as soon as the agent's current call is terminated.

Service Parameters:

device [mandatory] Must be a valid agent extension or a logical agent ID.

Private Parameters:

split [optional] If specified, it must be a valid ACD split extension. This parameter is optional in an EAS environment, but it is mandatory for a non-EAS environment.

Ack Parameters:

agentState [mandatory - partially supported] The ACD agent state. Agent state will be one of the following values:

- AG_NULL - The agent is logged out of the device/ACD split.
- AG_NOT_READY - The agent is occupied with some task other than that of serving a call.
- AG_WORK_NOT_READY - The agent is occupied with after call work. The agent should not receive additional ACD calls in this state.
- AG_READY - The agent is available to accept calls or is currently busy with an ACD call.

The DEFINITY PBX does not support the AG_WORK_READY state.

Ack Private Parameters:

workMode [optional] This parameter provides the agent work mode as defined by the DEFINITY PBX. Valid values include:

- WM_AUTO_IN Indicates that the agent is allowed to receive a new call immediately after disconnecting from the previous call. The talkState parameter indicates whether the agent is busy or idle.
- WM_MANUAL_IN Indicates that the agent is automatically changed to the WM_AFTCAL_WK state immediately after disconnecting from the previous call.
- WM_AFTCAL_WK Indicates that the agent is in the WM_AFTCAL_WK mode. (A query agent state on an agent in the WM_AFTCAL_WK state returns an agentState parameter value of AG_WORK_NOT_READY.)

- WM_AUX_WORK Indicates that the agent is in the WM_AUX_WORK mode. (A query agent state on an agent in the WM_AUX_WORK state returns an agentState parameter value of AG_NOT_READY.)
- talkState** [optional] The talkState parameter provides the actual readiness of the agent. Valid values are:
- TS_ON_CALL Indicates that the agent is occupied with serving a call
 - TS_IDLE Indicates that the agent is ready to accept calls.
- reasonCode** [optional] Specifies the reason code for the appropriate agent state. Valid reason codes are a single digit 1- 9. A value of 0 indicates that the reason code is not available. The meaning of the code (1-9) is defined by the application. This parameter is supported by private data version 5 and later only.
- pendingWorkMode** [optional] Specifies the work mode which will take effect when the agent gets off the call. If no work mode is pending then pendingWorkMode will be set to WM_NONE (-1).
- pendingReasonCode** [optional] Specifies the pending reason code which will take effect when the agent gets off the call. A value of 0 indicates that the pending reason code is not available.

Nak Parameter:

- universalFailure** If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error value, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:
- INVALID_CSTA_DEVICE_IDENTIFIER (12) An invalid device identifier has been specified in device.

Detailed Information:

- G3 PBX does not support the AG_WORK_READY state for agentState.
- Except agentState of AG_NULL, all confirmation includes private parameters of agent workMode and talkState. The actual readiness of the agent depends on values for these private parameters. In particular, the value for talkState determines if the agent is busy on a call or ready to accept calls.
- The G3 PBX Agent Work Mode to CSTA Agent State Mapping is shown in Table 6-1.

Table 6-1. G3 PBX Agent Work Mode Mapped to CSTA Agent State

G3 PBX Agent Work Mode	CSTA Agent State (workMode)
Agent not logged in	NULL
WM_AUX_WORK	AG_NOT_READY
WM_AFTCAL_WORK	AG_WORK_NOT_READY
WM_AUTO_IN	AG_READY (workMode=WM_AUTO_IN)
WM_MANUAL_IN	AG_READY (workMode=WM_MANUAL_IN)

- If the agent workMode is WM_AUTO_IN, the Query Agent State service always returns AG_READY. The agent is immediately made available to receive a new call after disconnecting from the previous call.

Table 6-2. Agent Activity Mapped to CSTA Agent State and Talk State

Agent Activity	agentState	talkState
Ready to accept calls	AG_READY	TS_IDLE
Occupied with a call	AG_READY	TS_ON_CALL
Disconnected from call	AG_READY	TS_IDLE

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaQueryAgentState() - Service Request

RetCode_t  cstaQueryAgentState (
    ACSHandle_tacsHandle,
    InvokeID_tinvokeID,
    DeviceID_t*device,
    PrivateData_t*privateData);

// CSTAQueryAgentStateConfEvent - Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_teventClass;// CSTACONFIRMATION
    EventType_teventType;// CSTA_QUERY_AGENT_STATE_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_teventHeader;
    union
    {
        struct
        {
            InvokeID_tinvokeID;
            union
            {
                CSTAQueryAgentStateConfEvent_t
                queryAgentState;
            }u;
        } cstaConfirmation;
    } event;
char  heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAQueryAgentStateConfEvent_t {
    AgentState_tagentState;
} CSTAQueryAgentStateConfEvent_t;

typedef enum AgentState_t {
    AG_NOT_READY = 0,
    AG_NULL = 1,
    AG_READY = 2,
    AG_WORK_NOT_READY = 3,
    AG_WORK_READY = 4// not used in G3 CSTA
} AgentState_t;

```

Private Data Version 6 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attQueryAgentState() - Service Request Private Data
Setup Function

RetCode_tattQueryAgentState (
    ATTPrivateData_t*privateData,
    DeviceID_t*split);

typedef struct ATTPrivateData_t
{
    char        vendor[32];
    unsigned shortlength;
    char        data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

// ATTQueryAgentStateConfEvent - Service Response Private
Data

typedef struct ATTEvent_t
{
    ATTEventType_teventType;// ATT_QUERY_AGENT_STATE_
CONF
    union
    {
        ATTQueryAgentStateConfEvent_tqueryAgentState;
    } u;
char heap[ATTPRIV_MAX_HEAP];
} ATTEvent_t;

typedef struct ATTQueryAgentStateConfEvent_t
{
    ATTWorkMode_tworkMode;// agent work mode
    ATTTalkState_ttalkState;// agent talk state
    long        reasonCode;// single digit 1-9
    ATTWorkMode_tpendingWorkMode;// pending agent work
mode
    long        pendingReasonCode;// single digit 1-9
} ATTQueryAgentStateConfEvent_t;

typedef enum ATTWorkMode_t
{
    WM_NONE = -1,    // No workmode is pending
    WM_AUX_WORK = 1,
    WM_AFTCAL_WK = 2,
```

Private Data Version 6 (Continued)

```
        WM_AUTO_IN = 3,  
        WM_MANUAL_IN = 4  
    } ATTWorkMode_t;  
  
typedef enum ATTTalkState_t  
{  
    TS_ON_CALL = 0,  
    TS_IDLE = 1  
} ATTTalkState_t;
```

Private Data Version 5 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attQueryAgentState() - Service Request Private Data Setup
Function

RetCode_tattQueryAgentState (
    ATTPrivateData_t*privateData,
    DeviceID_t*split);

typedef struct ATTPrivateData_t
{
    char          vendor[32];
    unsigned shortlength;
    char          data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

// ATTV5QueryAgentStateConfEvent - Service Response Private Data

typedef struct ATTEvent_t
{
    ATTEventType_teventType;// ATTV5_QUERY_AGENT_STATE_CONF
    union
    {
        ATTV5QueryAgentStateConfEvent_tv5queryAgentState;
    } u;
    char  heap[ATTPRIV_MAX_HEAP];
} ATTEvent_t;

typedef struct ATTV5QueryAgentStateConfEvent_t
{
    ATTWorkMode_tworkMode;// agent work mode
    ATTTalkState_ttalkState;// agent talk state
    long          reasonCode;// single digit 1-9
} ATTV5QueryAgentStateConfEvent_t;

typedef enum ATTWorkMode_t
{
    WM_AUX_WORK = 1,
    WM_AFTCAL_WK = 2,
    WM_AUTO_IN = 3,
    WM_MANUAL_IN = 4
} ATTWorkMode_t;

typedef enum ATTTalkState_t
{
    TS_ON_CALL = 0,
    TS_IDLE = 1
} ATTTalkState_t;
```


Private Data Versions 2-4 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attQueryAgentState() - Service Request Private Data Setup
Function

RetCode_tattQueryAgentState (
    ATTPrivateData_t*privateData,
    DeviceID_t*split);

typedef struct ATTPrivateData_t
{
    char        vendor[32];
    unsigned shortlength;
    char        data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

// ATTV4QueryAgentStateConfEvent - Service Response Private Data

typedef struct ATTEvent_t
{
    ATTEventType_teventType;// ATTV4_QUERY_AGENT_STATE_CONF
    union
    {
        ATTV4QueryAgentStateConfEvent_tv4queryAgentState;
    } u;
    char    heap[ATTPRIV_MAX_HEAP];
} ATTEvent_t;

typedef struct ATTV4QueryAgentStateConfEvent_t
{
    ATTWorkMode_tworkMode;// agent work mode
    ATTTalkState_ttalkState;// agent talk state
} ATTV4QueryAgentStateConfEvent_t;

typedef enum ATTWorkMode_t
{
    WM_AUX_WORK = 1,
    WM_AFTCAL_WK = 2,
    WM_AUTO_IN = 3,
    WM_MANUAL_IN = 4
} ATTWorkMode_t;

typedef enum ATTTalkState_t
{
    TS_ON_CALL = 0,
    TS_IDLE = 1
} ATTTalkState_t;
```

Query Call Classifier Service

Direction: Client to Switch Function: `cstaEscapeService()`
Confirmation Event: `CSTAEscapeServiceConfEvent`
Private Data Function: `attQueryCallClassifier()`
Private Data Confirmation Event: `ATTQueryCallClassifierConfEvent`
Service Parameters: `noData`
Private Parameters: `noData`
Ack Parameters: `noData`
Ack Private Parameters: `numAvailPorts, numInUsePorts`
Nak Parameter: `universalFailure`

Functional Description:

This service provides the number of "idle" and "in-use" ports (e.g., TN744). The "in use" number is a snapshot of the call classifier port usage.

Service Parameters:

noData None for this service.

Private Parameters:

noData None for this service.

Ack Parameters:

noData None for this service.

Ack Private Parameters:

numAvailPorts [mandatory] The number of available ports.

numInUsePorts [mandatory] The number of "in use" ports.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a `CSTAUniversalFailureConfEvent`. The error parameter in this event may be one of the error values described in the "CSTAUniversalFailureConfEvent" section in Chapter 3:

Detailed Information:

None

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaEscapeService() - Service Request

RetCode_t  cstaEscapeService (
    ACSHandle_tacsHandle,
    InvokeID_tinvokeID,
    PrivateData_t*privateData);

// CSTAEscapeServiceConfEvent - Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_teventClass;// CSTACONFIRMATION
    EventType_teventType;// CSTA_ESCAPE_SERVICE_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_teventHeader;
    union
    {
        struct
        {
            InvokeID_tinvokeID;
            union
            {
                CSTAEscapeSvcConfEvent_tescapeService;
            }u;
        } cstaConfirmation;
    } event;
    char  heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAEscapeSvcConfEvent_t {
    Nulltypenull
} CSTAEscapeSvcConfEvent_t;
```

Private Parameter Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attQueryCallClassifier() - Service Request Private Data
Setup Function

RetCode_tattQueryCallClassifier (// no private parameter,
// but must be called
    ATTPrivateData_t*privateData);

typedef struct ATTPrivateData_t {
    char    vendor[32];
    ushortlength;
    char    data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

// ATTQueryCallClassifierConfEvent - Service Response
Private Data

typedef struct
{
    ATTEventType_teventType;// ATT_QUERY_CALL_
CLASSIFIER_CONF
    union
    {
        ATTQueryCallClassifierConfEvent_t
queryCallClassifier;
    }u;
    char  heap[ATTPRIV_MAX_HEAP];
} ATTEvent_t;

typedef struct ATTQueryCallClassifierConfEvent_t
{
    shortnumAvailPorts;// number of available ports
    shortnumInUsePorts;// number of in use ports
} ATTQueryCallClassifierConfEvent_t;
```

Query Device Info

Direction: Client to Switch Function: cstaQueryDeviceInfo()

Confirmation Event: CSTAQueryDeviceInfoConfEvent

Private Data Confirmation Event: ATTQueryDeviceInfoConfEvent (private data version 5), ATTV4QueryDeviceInfoConfEvent (private data versions 2-4)

Service Parameters: device

Ack Parameters: device, deviceType, deviceClass

Ack Private Parameters: extensionClass, associatedDevice, associatedClass

Nak Parameter: universalFailure

Functional Description:

This service provides the class and type of a device. The class is one of the following attributes: voice, data, image, or other. The type is one of the following attributes: station, ACD, ACD Group, or other. The G3 Extension class is provided in the CSTA private data.

Service Parameters:

device [mandatory] Must be a valid on-PBX station extension.

Ack Parameters:

device [optional - supported] Normally this is the same ID specified in the device parameter in the request. See associatedDevice and associatedClass below.

deviceType [mandatory] The device type (mapped from G3 extension class).

deviceClass [mandatory] The device class (mapped from G3 extension class).

Ack Private Parameters:

extensionClass [mandatory] The G3 Extension Class for the device.

associatedDevice [optional] If the device specified in the request is a physical device of a logical agent who is logged in, the logical ID of that agent is returned in this parameter. Vice versa, if the device specified in the request is the logical ID of a logged-in agent, the physical device ID of that agent is returned in this parameter. Otherwise, a null string is returned. This parameter is supported by private data version 5 and later only.

associatedClass [optional] The G3 Extension Class for the associatedDevice. It is EC_LOGICAL_AGENT, if the associatedDevice is a device ID of a logical agent; otherwise it has the value of EC_OTHER. This parameter is supported by private data version 5 and later only.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error value, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- INVALID_CSTA_DEVICE_IDENTIFIER (12) An invalid device identifier has been specified in device.

Detailed Information:

The deviceType and deviceClass parameters are mapped from the G3 extension class as shown in Table 6-3.

Table 6-3. G3 Extension Class Mapping to Device Class and Type

G3 Extension Class	CSTA Device Class	CSTA Device Type
VDN	Voice ¹	ACD Group
Hunt Group (ACD Split)	Voice	ACD Group
Announcement	Voice	Other
Data extension	Data	Station
Voice extension — Analog	Voice	Station
Voice extension — Proprietary	Voice	Station
Voice extension — BRI	Voice	Station
Logical Agent	Voice	Other
CTI	Data	Other
Other (modem pool, etc.)	Other	Other

1. There is an additional private data qualifier that indicates if it is a VDN.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaQueryDeviceInfo() - Service Request

RetCode_t  cstaQueryDeviceInfo (
    ACSHandle_tacsHandle,
    InvokeID_tinvokeID,
    DeviceID_t*device,
    PrivateData_t*privateData);

// CSTAQueryDeviceInfoConfEvent - Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_teventClass;// CSTACONFIRMATION
    EventType_teventType;// CSTA_QUERY_DEVICE_INFO_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_teventHeader;
    union
    {
        struct
        {
            InvokeID_tinvokeID;
            union
            {
                CSTAQueryDeviceInfoConfEvent_t
                queryDeviceInfo;
            }u;
        } cstaConfirmation;
    } event;
    char  heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAQueryDeviceInfoConfEvent_t {
    DeviceID_tdevice;
    DeviceType_tdeviceType;
    DeviceClass_tdeviceClass;
} CSTAQueryDeviceInfoConfEvent_t;
```


Syntax (Continued)

```
// Device Types
typedef enum DeviceType_t {
    DT_STATION = 0,
    DT_LINE = 1, // not an expected G3 response
    DT_BUTTON = 2, // not an expected G3 response
    DT_ACD = 3,
    DT_TRUNK = 4, // not an expected G3 response
    DT_OPERATOR = 5, // not an expected G3 response
    DT_STATION_GROUP = 16, // not an expected G3
response
    DT_LINE_GROUP = 17, // not an expected G3 response
    DT_BUTTON_GROUP = 18, // not an expected G3 response
    DT_ACD_GROUP = 19,
    DT_TRUNK_GROUP = 20, // not an expected G3 response
    DT_OPERATOR_GROUP = 21, // not an expected G3
response
    DT_OTHER = 255
} DeviceType_t;

typedef unsigned char DeviceClass_t;

// Device Classes
#define DC_VOICE0x80
#define DC_DATA0x40
#define DC_IMAGE0x20 // not an expected G3
// response
#define DC_OTHER0x10
```

Private Data Version 5 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTQueryDeviceInfoConfEvent - Service Response Private
Data

typedef struct ATTEvent_t
{
    ATTEventType_teventType;// ATT_QUERY_DEVICE_INFO_
CONF
    union
    {
        ATTQueryDeviceInfoConfEvent_tqueryDeviceInfo;
    } u;
char heap[ATTPRIV_MAX_HEAP];
} ATTEvent_t;

typedef struct ATTQueryDeviceInfoConfEvent_t
{
    ATTExtensionClass_textensionClass;
    ATTExtensionClass_tassociatedClass;
    DeviceID_t associatedDevice;
} ATTQueryDeviceInfoConfEvent_t;

typedef enum ATTExtensionClass_t
{
    EC_VDN = 0,
    EC_ACD_SPLIT = 1,
    EC_ANNOUNCEMENT = 2,
    EC_DATA = 4,
    EC_ANALOG = 5,
    EC_PROPRIETARY = 6,
    EC_BRI = 7,
    EC_CTI = 8,
    EC_LOGICAL_AGENT= 9,
    EC_OTHER = 10
} ATTExtensionClass_t;
```

Private Data Versions 2-4 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV4QueryDeviceInfoConfEvent - Service Response
Private Data

typedef struct ATTEvent_t
{
    ATTEventType_teventType;// ATTV4_QUERY_DEVICE_INFO_
CONF
    union
    {
        ATTV4QueryDeviceInfoConfEvent_t
v4queryDeviceInfo;
    } u;
char heap[ATTPRIV_MAX_HEAP];
} ATTEvent_t;

typedef struct ATTV4QueryDeviceInfoConfEvent_t
{
    ATTExtensionClass_textensionClass;
} ATTV4QueryDeviceInfoConfEvent_t;

typedef enum ATTExtensionClass_t
{
    EC_VDN = 0,
    EC_ACD_SPLIT = 1,
    EC_ANNOUNCEMENT = 2,
    EC_DATA = 4,
    EC_ANALOG = 5,
    EC_PROPRIETARY = 6,
    EC_BRI = 7,
    EC_CTI = 8,
    EC_LOGICAL_AGENT= 9,
    EC_OTHER = 10
} ATTExtensionClass_t;
```

Query Device Name Service

Direction: Client to Switch Function: cstaEscapeService()

Confirmation Event: CSTAEscapeSvcConfEvent

Private Data Function: attQueryDeviceName()

Private Data Confirmation Event: ATTQueryDeviceNameConfEvent (private data version 5), ATTV4QueryDeviceNameConfEvent (private data versions 2-4)

Service Parameters: noData

Private Parameters: device

Ack Parameters: noData

Ack Private Parameters: deviceType, device, name, unname

Nak Parameter: universalFailure

Functional Description:

The Query Device Name service allows an application to query the switch with an extension of a device and receive the associated name of the device. The name is retrieved from the G3 PBX Integrated Directory Database.

This service will allow an application to identify the names administered in the G3 switch with device extension numbers without maintaining its own database.

Service Parameters:

noData None for this service.

Private Parameters:

device [mandatory] Must be a valid device extension.

Ack Parameters:

noData None for this service.

Ack Private Parameters:

deviceType [mandatory] Specifies the device type of the device:

- DT_ACD_SPLIT — ACD Split (Hunt Group)
- DT_ANNOUNCEMENT — announcement
- DT_DATA — data extension
- DT_LOGICAL_AGENT — logical agent
- DT_STATION — station extension
- DT_TRUNK_ACCESS_CODE — Trunk Access Code
- DT_VDN — VDN

⇒ NOTE:

If no name is administered in the G3 switch for the device, the *deviceType* and the *name* parameters will not contain meaningful information. Also the *deviceType* is set to 0 (0 is not defined in the list above).

device [mandatory] Specifies the extension number of the device.

name [mandatory] Specifies the associated name of the device. This is a string of 1-15 ASCII characters for private data version 3 and 4. This is a string of 1-27 ASCII characters for private data version 5 and later only.

⇒ NOTE:

The name of a device is administered in the G3 PBX. Non-standard 8-bit OPTREX characters supported on the displays of the 84xx series terminals may be reported in name parameter. The 84xx terminal displays supports a limited number of non-standard characters (in addition to the standard 7-bit ASCII display characters), including Katakana, graphical characters, and Eurofont (European-type) characters. The tilde,~, character is not defined in the OPTREX set and is used as the toggle character (turn on/off 8-bit character set) to indicate subsequent characters are to have the high-bit set (turned off by a following ~ character, if any). If non-standard 8-bit OPTREX characters are administered in the switch for the device, then the tilde,~, character will be reported in its name. An application needs to map the non-standard 8-bit OPTREX characters to its proper printable characters.

For additional information, see Appendix A, "Enhanced Voice Terminal Display."

uname

[mandatory] Specifies the associated name of the device in Unicode . This parameter is supported by private data version 5 and later only.

Nak Parameter:

- universalFailure*** If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:
- VALUE_OUT_OF_RANGE (3) (CS0/100) Invalid parameter value specified.
 - OBJECT_NOT_KNOWN (4) (CS0/96) Mandatory parameter is missing.
 - INVALID_CSTA_DEVICE_IDENTIFIER (12) (CS0/28) An invalid device identifier has been specified in device.
 - GENERIC_SUBSCRIBED_RESOURCE_AVAILABILITY (41) (CS0/50) This service is requested on a G3 PBX administered as a release earlier than G3V4.

Detailed Information:

- Incomplete Names — The names returned by this service may not be the full names since they are limited to 15 characters in the Integrated Directory database.
- Security — G3 Switch does not provide security mechanisms for this service.
- Traffic Control — The application is responsible for controlling the message traffic on the CTI link. An application should minimize traffic by requesting device names only when needed. This service is not intended for use by an application to create its own copy of the Integrated Directory database. If the number of outstanding requests reaches the switch limit, the response time may be as long as 30 seconds.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaEscapeService() - Service Request

RetCode_t  cstaEscapeService (
    ACSHandle_tacsHandle,
    InvokeID_tinvokeID,
    PrivateData_t*privateData);

// CSTAEscapeSvcConfEvent - Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_teventClass;// CSTACONFIRMATION
    EventType_teventType;// CSTA_ESCAPE_SVC_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_teventHeader;
    union
    {
        struct
        {
            InvokeID_tinvokeID;
            union
            {
                CSTAEscapeSvcConfEventescapeService;
            }u;
        } cstaConfirmation;
    } event;
    char  heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAEscapeSvcConfEvent_t {
    Nulltypenull
} CSTAEscapeSvcConfEvent_t;
```


Private Data Version 5 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attQueryDeviceName() - Service Request Private Data
Setup Function

RetCode_t attQueryDeviceName (
    ATTPrivateData_t*privateData,
    DeviceID_t*device);

typedef struct ATTPrivateData_t
{
    char    vendor[32];
    ushort length;
    char    data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

// ATTQueryDeviceNameConfEvent - Service Response Private
Data

typedef struct ATTEvent_t
{
    ATTEventType_t eventType; // ATT_QUERY_DEVICE_NAME_
CONF
    union
    {
        ATTQueryDeviceNameConfEvent_t queryDeviceName;
    } u;
    char heap[ATTPRIV_MAX_HEAP];
} ATTEvent_t;

typedef struct ATTQueryDeviceNameConfEvent_t
{
    ATTDeviceType_t deviceType;
    DeviceID_t device;
    DeviceID_t name; // 1-27 ASCII character string
    ATTUnicodeDeviceID_t unicodeName; // name in Unicode
} ATTQueryDeviceNameConfEvent_t;

typedef enum ATTDeviceType_t
{
    ATT_DT_ACD_SPLIT= 1,
    ATT_DT_ANNOUNCEMENT= 2,
    ATT_DT_DATA = 3,
    ATT_DT_LOGICAL_AGENT= 4,
    ATT_DT_STATION= 5,
```

```
        ATT_DT_TRUNK_ACCESS_CODE= 6 ,  
        ATT_DT_VDN           = 7  
    }ATTDeviceType_t;
```

Private Data Versions 2-4 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attQueryDeviceName() - Service Request Private Data
Setup Function

RetCode_t attQueryDeviceName (
    ATTPrivateData_t*privateData,
    DeviceID_t*device);

typedef struct ATTPrivateData_t
{
    char    vendor[32];
    ushort length;
    char    data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

// ATTV4QueryDeviceNameConfEvent - Service Response
Private Data

typedef struct ATTEvent_t
{
    ATTEventType_t eventType; // ATTV4_QUERY_DEVICE_NAME_
CONF
    union
    {
        ATTV4QueryDeviceNameConfEvent_t
v4queryDeviceName;
    } u;
    char heap[ATTPRIV_MAX_HEAP];
} ATTEvent_t;

typedef struct ATTV4QueryDeviceNameConfEvent_t
{
    ATTDeviceType_t deviceType;
    DeviceID_t device;
    char    name[16]; // 1-15 ASCII character string
} ATTV4QueryDeviceNameConfEvent_t;

typedef enum ATTDeviceType_t
{
    ATT_DT_ACD_SPLIT= 1,
    ATT_DT_ANNOUNCEMENT= 2,
    ATT_DT_DATA      = 3,
    ATT_DT_LOGICAL_AGENT= 4,
    ATT_DT_STATION= 5,
```

```
        ATT_DT_TRUNK_ACCESS_CODE= 6 ,  
        ATT_DT_VDN          = 7  
    } ATTDeviceType_t;
```

Query Do Not Disturb Service

Direction: Client to Switch Function: `cstaQueryDoNotDisturb()`

Confirmation Event: `CSTAQueryDoNotDisturbConfEvent`

Service Parameters: `device`

Ack Parameters: `doNotDisturb`

Nak Parameter: `universalFailure`

Functional Description:

This service provides the status of the send all calls feature expressed as on or off at a device. The status will always be reported as off when the extension does not have a coverage path.

Service Parameters:

device [mandatory] Must be a valid on-PBX station extension that supports the send all calls (SAC) feature..

Ack Parameters:

doNotDisturb [mandatory] Status of the send all calls feature expressed as on (TRUE) or off (FALSE).

Nak Parameter:

universalFailure If the request is not successful, the application will receive a `CSTAUniversalFailureConfEvent`. The error parameter in this event may contain the following error value, or one of the error values described in the “`CSTAUniversalFailureConfEvent`” section in Chapter 3:

- `INVALID_CSTA_DEVICE_IDENTIFIER` (12) An invalid device identifier has been specified in `device`.

Detailed Information:

None

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaQueryDoNotDisturb() - Service Request

RetCode_t  cstaQueryDoNotDisturb (
    ACSHandle_tacsHandle,
    InvokeID_tinvokeID,
    DeviceID_t*device,
    PrivateData_t*privateData);

// CSTAQueryDoNotDisturbConfEvent - Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_teventClass;
    EventType_teventType;
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_teventHeader;
    union
    {
        struct
        {
            InvokeID_tinvokeID;
            union
            {
                CSTAQueryDndConfEvent_tqueryDnd;
            }u;
        } cstaConfirmation;
    } event;
    char  heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAQueryDndConfEvent_t {
    Boolean_tdoNotDisturb;// TRUE = on, FALSE = off
} CSTAQueryDndConfEvent_t;
```

Query Forwarding Service

Direction: Client to Switch Function: `cstaQueryForwarding()`

Confirmation Event: `CSTAQueryForwardingConfEvent`

Service Parameters: `device`

Ack Parameters: `forward`

Nak Parameter: `universalFailure`

Functional Description:

This service provides the status and forward-to-number of the Call Forwarding feature for a device. The status is expressed as on or off. The G3 PBX only supports one Forwarding Type (Immediate). Thus, the on/off indicator is only specified for the Immediate type. The Call Forwarding feature may be turned on for many types (G3 redirection Criteria), and the actual forward type is dependent on how the feature is administered in the G3 PBX.

Service Parameters:

device [mandatory] Must be a valid on-PBX station extension that supports the Call Forwarding feature.

Ack Parameters:

forward [mandatory] This is a list of forwarding parameters. The list contains a count of how many items are in the list. Since the G3 PBX only stores one forwarding address, the count is one. Each element in the list contains the following: forwardingType, forwardingOn, and forwardDN. For G3 PBX, forwardingType will always be FWD_IMMEDIATE; forwardingOn will indicate (on/off) status (TRUE indicates on, FALSE indicates off); and forwardDN will contain the forward-to-number.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a `CSTAUniversalFailureConfEvent`. The error parameter in this event may contain the following error value, or one of the error values described in the “`CSTAUniversalFailureConfEvent`” section in Chapter 3:

- `INVALID_CSTA_DEVICE_IDENTIFIER` (12) An invalid device identifier has been specified in device.

Detailed Information:

The G3 PBX supports only one CSTA Forwarding Type: Immediate. Thus, each response contains information for the Immediate type.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaQueryForwarding() - Service Request

RetCode_t  cstaQueryForwarding (
    ACSHandle_tacsHandle,
    InvokeID_tinvokeID,
    DeviceID_t*deviceID,
    PrivateData_t*privateData);

// CSTAQueryForwardingConfEvent - Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_teventClass;// CSTACONFIRMATION
    EventType_teventType;// CSTA_QUERY_FWD_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_teventHeader;
    union
    {
        struct
        {
            InvokeID_tinvokeID;
            union
            {
                CSTAQueryFwdConfEvent_tqueryFwd;
            }u;
        } cstaConfirmation;
    } event;
    char  heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAQueryFwdConfEvent_t {
    ListForwardParameters_tforward;
} CSTAQueryFwdConfEvent_t;
```


Syntax (Continued)

```
typedef struct ListForwardParameters_t {
    shortcount;          // only 1 is provided in list
    ForwardingInfo_tparam[7];
} ListForwardParameters_t;

typedef struct ForwardingInfo_t {
    ForwardingType_tforwardingType; // FWD_IMMEDIATE
    Boolean forwardingOn;// TRUE = on, FALSE = off
    DeviceID_tforwardDN;
} ForwardingInfo_t;

typedef enum ForwardingType_t {
    FWD_IMMEDIATE = 0, // only type supported
    FWD_BUSY = 1, // not supported
    FWD_NO_ANS = 2, // not supported
    FWD_BUSY_INT = 3, // not supported
    FWD_BUSY_EXT = 4, // not supported
    FWD_NO_ANS_INT = 5, // not supported
    FWD_NO_ANS_EXT = 6 // not supported
} ForwardingType_t;
```

Query Message Waiting Service

Direction: Client to Switch **Function:** `cstaQueryMsgWaitingInd()`

Confirmation Event: `CSTAQueryMwiConfEvent`

Private Data Confirmation Event: `ATTQueryMwiConfEvent`

Service Parameters: `device`

Ack Parameters: `messages`

Ack Private Parameters: `applicationType`

Nak Parameter: `universalFailure`

Functional Description:

The Query Message Waiting Service provides status of the message waiting indicator expressed as on or off for a device. The applications that turn the indicator on (that is, ASAI, Property Management, Message Center, Voice Processing, Leave Word Calling) are reported in the private data.

Service Parameters:

device [mandatory] Must be a valid on-PBX station extension that supports the Message Waiting Indicator (MWI) feature.

Ack Parameters:

messages [mandatory] Indicates the on/off status (TRUE indicates on, FALSE indicates off) of the MWI for this device.

Ack Private Parameters:

applicationType [mandatory] Indicates the applications that turned on the MWI for the device

Nak Parameter:

universalFailure If the request is not successful, the application will receive a `CSTAUniversalFailureConfEvent`. The error parameter in this event may contain the following error value, or one of the error values described in the “`CSTAUniversalFailureConfEvent`” section in Chapter 3:

- `INVALID_CSTA_DEVICE_IDENTIFIER` (12) An invalid device identifier has been specified in `device`.

Detailed Information:

- Application Type — The private data member applicationType is a bit map where one bit is set for each application that turned on the indicator. Multiple applications may turn on the indicator. The applications represented are: CTI/ASAI, Property Management (PMS), Message Center (MCS), Voice Messaging, and Leave Word Calling (LWC).

To find out which applications turned on the indicator, the application must use a bit mask as shown in Table 6-4:

Table 6-4. Application Types Mapped to Bit Maps

bit:	8	7	6	5	4	3	2	1
Application	N/A	N/A	N/A	CTI/ASAI	LWC	PMS	Voice	MCS

- Setting MWI Status — An application can set the MWI status through the CSTASetMsgWaitingInd Service.
- System Starts — System cold starts cause the switch to lose the MWI status. Other types of restart do not affect the MWI status.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaQueryMsgWaitingInd() - Service Request

RetCode_t  cstaQueryMsgWaitingInd (
    ACSHandle_tacsHandle,
    InvokeID_tinvokeID,
    DeviceID_t*device,
    PrivateData_t*privateData);

// CSTAQueryMsgWaitingIndConfEvent - Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_teventClass;// CSTACONFIRMATION
    EventType_teventType;// CSTA_QUERY_MWI_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_teventHeader;
    union
    {
        struct
        {
            InvokeID_tinvokeID;
            union
            {
                CSTAQueryMwiConfEvent_tqueryMwi;
            }u;
        } cstaConfirmation;
    } event;
    char  heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAQueryMwiConfEvent_t {
    Booleanmessages;// TRUE = on, FALSE = off
} CSTAQueryMwiConfEvent_t;
```

Private Parameter Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTQueryMwiConfEvent - Service Response Private Data

typedef struct
{
    ATTEventTypeeventType;// ATT_QUERY_MWI_CONF
    union
    {
        ATTQueryMwiConfEvent_tqueryMwi;
    }u;
    char heap[ATTPRIV_MAX_HEAP];
} ATTEvent_t;

typedef struct ATTQueryMwiConfEvent_t
{
    ATTMwiApplication_tapplicationType;// application
    type
} ATTQueryMwiConfEvent_t;

typedefunsigned charATTMwiApplication_t;
#define AT_MCS 0x01// bit 1
#define AT_VOICE0x02// bit 2
#define AT_PROPMGT0x04 // bit 3
#define AT_LWC 0x08// bit 4
#define AT_CTI 0x10// bit 5
```

Query Station Status Service

Direction: Client to Switch
Function: `cstaEscapeService()`
Confirmation Event: `CSTAEscapeServiceConfEvent`
Private Data Function: `attQueryStationStatus()`
Private Data Confirmation Event: `ATTQueryStationStatusConfEvent`
Service Parameters: `noData`
Private Parameters: `device`
Ack Parameters: `noData`
Ack Private Parameters: `stationStatus`
Nak Parameter: `universalFailure`

Functional Description:

The Query Station Status service provides the idle and/or busy state of a station. The "busy" state is returned if the station is active with a call. The "idle" state is returned if the station is not active with any call.

Service Parameters:

noData None for this service.

Private Parameters:

device [mandatory] Must be a valid station device.

Ack Parameters:

noData None for this service.

Ack Private Parameters:

stationStatus [mandatory] Specifies the busy/idle state (TRUE indicates busy, FALSE indicates idle) of the station.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a `CSTAUniversalFailureConfEvent`. The error parameter in this event may contain the following error value, or one of the error values described in the "CSTAUniversalFailureConfEvent" section in Chapter 3:

- `INVALID_CSTA_DEVICE_IDENTIFIER (12)` An invalid device identifier has been specified in `device`.

Detailed Information:

None

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaEscapeService() - Service Request

RetCode_t  cstaEscapeService (
    ACSHandle_tacsHandle,
    InvokeID_tinvokeID,
    PrivateData_t*privateData);

// CSTAEscapeServiceConfEvent - Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_teventClass;// CSTACONFIRMATION
    EventType_teventType;// CSTA_ESCAPE_SERVICE_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_teventHeader;
    union
    {
        struct
        {
            InvokeID_tinvokeID;
            union
            {
                CSTAEscapeSvcConfEvent_tescapeService;
            }u;
        } cstaConfirmation;
    } event;
    char  heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAEscapeSvcConfEvent_t {
    Nulltypenull
} CSTAEscapeSvcConfEvent_t;
```

Private Parameter Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attQueryStationStatus() - Service Request Private Data
Setup Function

RetCode_tattQueryStationStatus (// returns NULL if no
// parameter specified
    ATTPrivateData_t*privateData,
    DeviceID_t*device);

typedef struct ATTPrivateData_t {
    char        vendor[32];
    unsigned shortlength;
    char        data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

// ATTQueryStationStatusConfEvent - Service Response
Private Data

typedef struct
{
    ATTEventTypeeventType;// ATT_QUERY_STATION_STATUS_
CONF
    union
    {
        ATTQueryStationStatusConfEvent_t
queryStationStatus;
    }u;
    charheap[ATTPRIV_MAX_HEAP];
} ATTEvent_t;

typedef struct ATTQueryStationStatusConfEvent_t
{
    BooleanstationStatus;// TRUE = busy, FALSE = idle
} ATTQueryStationStatusConfEvent_t;
```


Query Time Of Day Service

Direction: Client to Switch **Function:** `cstaEscapeService()`
Confirmation Event: `CSTAEscapeServiceConfEvent`
Private Data Function: `attQueryTimeOfDay()`
Private Data Confirmation Event: `ATTQueryTimeofDayConfEvent`
Service Parameters: `noData`
Private Parameters: `noData`
Ack Parameters: `noData`
Ack Private Parameters: `time`
Nak Parameter: `universalFailure`

Functional Description:

The Query Time of Day Service provides the switch information for the year, month, day, hour, minute, and second.

Service Parameters:

noData None for this service.

Ack Parameters:

noData None for this service.

Ack Private Parameters:

time [mandatory] Specifies the year, month, day, hour, minute, and second.

- The year 1999 is specified by two digits — 99.
- The year 2000 is specified by one digit — 0.
- The year 2001 is specified by one digit — 1.
- The year 2002 is specified by one digit — 2, and so forth.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a `CSTAUniversalFailureConfEvent`. The error parameter in this event may contain one of the error values described in the “`CSTAUniversalFailureConfEvent`” section in Chapter 3:

Detailed Information:

None

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaEscapeService() - Service Request

RetCode_t  cstaEscapeService (
    ACSHandle_tacsHandle,
    InvokeID_tinvokeID,
    PrivateData_t*privateData);

// CSTAEscapeServiceConfEvent - Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_teventClass;// CSTACONFIRMATION
    EventType_teventType;// CSTA_ESCAPE_SERVICE_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_teventHeader;
    union
    {
        struct
        {
            InvokeID_tinvokeID;
            union
            {
                CSTAEscapeSvcConfEvent_tescapeService;
            }u;
        } cstaConfirmation;
    } event;
    char  heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAEscapeSvcConfEvent_t {
    Nulltypenull
} CSTAEscapeSvcConfEvent_t;
```

Private Parameter Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attQueryTimeOfDay() - Service Request Private Data
Setup Function

RetCode_tattQueryTimeOfDay (// no private parameter,
// but must be called
    ATTPrivateData_t*privateData);

typedef struct ATTPrivateData_t {
    char        vendor[32];
    unsigned shortlength;
    char        data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

// ATTQueryTimeOfDayConfEvent - Service Response Private
Data

typedef struct
{
    ATTEventTypeeventType;// ATT_QUERY_TOD_CONF
    union
    {
        ATTQueryTODConfEvent_tqueryTOD;
    }u;
    char heap[ATTPRIV_MAX_HEAP];
} ATTEvent_t;

typedef struct ATTQueryTODConfEvent_t
{
    shortyear;
    shortmonth;
    shortday;
    shorthour;
    shortminute;
    shortsecond;
} ATTQueryTODConfEvent_t;
```

Query Trunk Group Service

Direction: Client to Switch
Function: `cstaEscapeService()`
Confirmation Event: `CSTAEscapeServiceConfEvent`
Private Data Function: `attQueryTrunkGroup()`
Private Data Confirmation Event: `ATTQueryTrunkGroupConfEvent`
Service Parameters: `noData`
Private Data Parameters: `device`
Ack Parameters: `noData`
Ack Private Parameters: `idleTrunks, usedTrunks`
Nak Parameter: `universalFailure`

Functional Description:

The Query Trunk Group Service provides the number of idle trunks and the number of in-use trunks. The sum of the idle and in-use trunks provides the number of trunks in service.

Service Parameters:

noData None for this service.

Private Data Parameters:

device [mandatory] Specifies a valid trunk group access code.

Ack Parameters:

noData None for this service.

Ack Private Parameters:

idleTrunks [mandatory] The number of "idle" trunks in the group.

usedTrunks [mandatory] The number of "in use" trunks in the group

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error value, or one of the error values described in the "CSTAUniversalFailureConfEvent" section in Chapter 3:

- INVALID_CSTA_DEVICE_IDENTIFIER (12) An invalid device identifier has been specified in device.

Detailed Information:

None

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaEscapeService() - Service Request

RetCode_t  cstaEscapeService (
    ACSHandle_tacsHandle,
    InvokeID_tinvokeID,
    PrivateData_t*privateData);

// CSTAEscapeServiceConfEvent - Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_teventClass;// CSTACONFIRMATION
    EventType_teventType;// CSTA_ESCAPE_SERVICE_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_teventHeader;
    union
    {
        struct
        {
            InvokeID_tinvokeID;
            union
            {
                CSTAEscapeSvcConfEvent_tescapeService;
            }u;
        } cstaConfirmation;
    } event;
    char  heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAEscapeSvcConfEvent_t {
    Nulltypenull
} CSTAEscapeSvcConfEvent_t;
```

Private Parameter Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attQueryTrunkGroup() - Service Request Private Data
Setup Function

RetCode_t attQueryTrunkGroup (// returns NULL if no
// parameter specified
    ATTPrivateData_t*privateData,
    DeviceID_t *device);

typedef struct ATTPrivateData_t {
    char        vendor[32];
    unsigned shortlength;
    char        data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

// ATTQueryTrunkGroupConfEvent - Service Response Private
Data

typedef struct
{
    ATTEventTypeeventType;// ATT_QUERY_TG_CONF
    union
    {
        ATTQueryTGConfEvent_tqueryTg;
    }u;
    char heap[ATTPRIV_MAX_HEAP];
} ATTEvent_t;

typedef struct ATTQueryTGConfEvent_t
{
    short    idleTrunks;// number of "idle" trunks
// in the group
    short    usedTrunks;// number of "in use" trunks
// in the group
} ATTQueryTGConfEvent_t;
```

Query Universal Call ID Service (Private)

Direction: Client to Switch Function: `cstaEscapeService()`
Confirmation Event: `CSTAEscapeServiceConfEvent`
Private Data Function: `attQueryUCID()`
Private Data Confirmation Event: `ATTQueryUCIDConfEvent`
Service Parameters: `noData`
Private Parameters: `call`
Ack Parameters: `noData`
Ack Private Parameters: `ucid`
Nak Parameter: `universalFailure`

Functional Description:

The Query Universal Call ID Service responds with the Universal Call ID (UCID) for a normal callID. This query may be requested to switch at anytime during the life of a call.

Service Parameters:

noData None for this service.

Private Parameters:

call [mandatory] Specifies the normal callID of a call. This is a Connection Identifier. The deviceID is ignored.

Ack Parameters:

noData None for this service.

Ack Private Parameters:

ucid [mandatory] Specifies the Universal Call ID (UCID) of the requested call. The UCID is a unique call identifier across switches and the network. A valid UCID is a null-terminated ASCII character string. If there is no UCID associated with this call, the ucid contains the ATT_NULL_UCID (a 20-character string of all zeros).

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- INVALID_CSTA_CALL_IDENTIFIER (11) (CS0/100, CS3/63) An invalid call identifier has been specified in call
- INVALID_FEATURE(15) (CS0/111) The switch software does not support this feature. The switch software release may be earlier than R6.

Detailed Information:

None

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaEscapeService() - Service Request

RetCode_t  cstaEscapeService (
    ACSHandle_tacsHandle,
    InvokeID_tinvokeID,
    PrivateData_t*privateData);

// CSTAEscapeServiceConfEvent - Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_teventClass;// CSTACONFIRMATION
    EventType_teventType;// CSTA_ESCAPE_SERVICE_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_teventHeader;
    union
    {
        struct
        {
            InvokeID_tinvokeID;
            union
            {
                CSTAEscapeSvcConfEvent_tescapeService;
            }u;
        } cstaConfirmation;
    } event;
    char  heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAEscapeSvcConfEvent_t {
    Nulltypenull
} CSTAEscapeSvcConfEvent_t;
```

Private Parameter Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attQueryUCID() - Service Request Private Data Setup
Function

RetCode_t attQueryUCID (
    ATTPrivateData_t *privateData,
    ConnectionID_t *call);

typedef struct ATTPrivateData_t {
    char          vendor[32];
    unsigned short length;
    char          data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

// ATTQueryUCIDConfEvent - Service Response Private Data

typedef struct
{
    ATTEventType eventType; // ATT_QUERY_UCID_CONF
    union
    {
        ATTQueryUCIDConfEvent_t queryUCID;
    } u;
    char heap[ATTPRIV_MAX_HEAP];
} ATTEvent_t;

typedef struct ATTQueryUCIDConfEvent_t
{
    ATTUCID_t ucid;
} ATTQueryUCIDConfEvent_t
```

Snapshot Service Group

7

Overview

This service group enables the client to “take a snapshot” of 1) information concerning a particular call and 2) information concerning calls associated with a particular device.

Centre Vu Computer Telephony (CVCT) supports the following CSTA Services:

- Snapshot Call Service
- Snapshot Device Service

Snapshot Call Service

Direction: Client to Switch

Function: *cstaSnapshotCallReq()*

Confirmation Event: *CSTASnapshotCallConfEvent*

Service Parameters: *snapshotObj*

Ack Parameters: *snapshotData*

Nak Parameter: *universalFailure*

Functional Description:

The Snapshot Call Service provides the following information for each endpoint on the specified call:

- Device ID
- Connection ID
- CSTA Local Connection State

The CSTA Connection state may be one of the following: Unknown, Null, Initiated, Alerting, Queued, Connected, Held, or Failed.

The Device ID may be an on-PBX extension, an alerting extension, or a split hunt group extension (when the call is queued). When a call is queued on more than one split hunt group, only one split hunt group extension is provided in the response to such a query. For calls alerting at various groups (for example, hunt group, TEG, etc.), the group extension is reported to the client application. For calls connected to a member of a group, the group member's extension is reported to the client.

Service Parameters:

snapshotObj [mandatory] Identifies the call object for which snapshot information is requested. The structure includes the call identifier, the device identifier, and the device type (static or dynamic).
The G3 PBX ignores the device identifier and device type, so they may have null values.

Ack Parameters:

snapshotData [mandatory] Contains all the snapshot information for the call for which the request was made. The structure includes a count of how many device endpoints are on the call as well as the following detailed information for each endpoint: Device ID, Call ID, and Local Connection State of the call at the device.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- INVALID_CSTA_CALL_IDENTIFIER (11) An invalid call identifier has been specified in snapshotObj.
- INVALID_CSTA_DEVICE_IDENTIFIER (12) An invalid device identifier has been specified in snapshotObj.
-

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaSnapshotCallReq() - Service Request

RetCode_t  cstaSnapshotCallReq
    ACSHandle_t    acsHandle,
    InvokeID_t     invokeID,
    ConnectionID_t *snapshotObj;
    PrivateData_t  *privateData);

// CSTASnapshotCallConfEvent - Service Response

typedef struct
{
    ACSHandle_t    acsHandle;
    EventClass_t   eventClass; // CSTACONFIRMATION
    EventType_t    eventType; // CSTA_SNAPSHOT_CALL_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t    eventHeader;
    union
    {
        struct
        {
            InvokeID_t    invokeID;
            union
            {
                CSTASnapshotCallConfEvent_t snapshotCall;
            }u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTASnapshotCallConfEvent_t {
    CSTASnapshotCallData_t    snapshotCall;
} CSTASnapshotCallConfEvent_t;

typedef struct CSTASnapshotCallData_t {
    int            count; // count of calls
    struct         CSTASnapshotCallResponseInfo_t    *info;
} CSTASnapshotCallData_t;

```


Syntax (Continued)

```
typedef struct CSTASnapshotCallResponseInfo_t {
    SubjectDeviceID_t      deviceOnCall;
    ConnectionID_t         callIdentifier;
    LocalConnectionState_t localConnectionState;
} CSTASnapshotCallResponseInfo_t;
```

Snapshot Device Service

Direction: Client to Switch

Function: *cstaSnapshotDeviceReq()*

Confirmation Event: *CSTASnapshotDeviceConfEvent*

Private Data Confirmation Event: *ATTSnapshotDeviceConfEvent* (private data version 5), *ATTV4SnapshotDeviceConfEvent* (private data versions 2-4)

Service Parameters: *snapshotObj*

Ack Parameters: *snapshotDevice*

Ack Private Parameters: *attSnapshotDevice*

Nak Parameter: *universalFailure*

Functional Description:

The Snapshot Device Service provides information about calls associated with a given CSTA device. The information identifies each call and indicates the CSTA local connection state for all devices on each call.

⇒ NOTE:

In the Release 2.0 product, the list of connection states for each call may not be a complete list.

Service Parameters::

snapshotObj [mandatory] Must be a valid device.

Ack Parameters:

snapshotDevice [mandatory] Contains a sequence of information about each call on the device. Information for each call includes the connectionID and a sequence of local connection states for each connection in the call.

Ack Private Parameters:

attsnapshotDevice [mandatory] Contains a sequence of information about each call on the device. Information for each call includes the connectionID and the G3 call state for each call at the snapshot device.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error value, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- INVALID_CSTA_DEVICE_IDENTIFIER (12) An invalid device identifier has been specified in device or forwardingDN.

Detailed Information:

- The ECMA-180 definition for the ack response does not distinguish between the call states for each individual connection making up a call. This is a deficiency because there is no way to correlate the local connection state to a particular connection ID within a call. To overcome this deficiency, the G3 PBX always returns the local connection state for the queried device first in the list for each of the calls. The response contains lists of connection states for each call at the snapshot device.
- Information for a maximum of 10 calls is provided for the snapshot device. This is a G3 PBX limit.
- Table 7-1 illustrates the mapping from the G3 PBX call state to the CSTA local call state (provided in the CSTA response):

Table 7-1. Mapping of G3 and CSTA Local Call States

G3 Local Call State	CSTA Local Call State
Initiate	Initiated
Alerting	Alerting
Connected	Connected
Held	Hold
Bridged	Null
Other	None (CS_NONE)

- The bridged state is a G3 PBX private local connection state that is not defined in the CSTA. This state indicates that a call is present at a bridged, simulated bridged, button TEG, or PCOL appearance, and the call is neither ringing nor connected at the station. The bridged connection state is reported in the private data of a Snapshot Device Confirmation Event and it has a CSTA null (CS_NULL) state. Thus a device with the null state in the Snapshot Device Confirmation Event is bridged.
- A device with the bridged state can join the call by manually answering the call (press the line appearance) or through the `cstaAnswerCall` service. Once a bridged device is connected to a call, its state becomes connected. After a bridged device becomes connected, it can drop from the call and become bridged again, if the call is not cleared.

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaSnapshotDeviceReq() - Service Request

RetCode_t    cstaSnapshotDeviceReq
    ACSHandle_t    acsHandle,
    InvokeID_t    invokeID,
    DeviceID_t    *snapshotObj;

// CSTASnapshotDeviceReqConfEvent - Service Response

typedef struct
{
    ACSHandle_t    acsHandle;
    EventClass_t    eventClass; // CSTACONFIRMATION
    EventType_t    eventType; // CSTA_SNAPSHOT_DEVICE_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t    eventHeader;
    union
    {
        struct
        {
            InvokeID_t    invokeID;
            union
            {
                CSTASnapshotDeviceConfEvent_t    snapshotDevice;
            }u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTASnapshotDeviceConfEvent_t {
    CSTASnapshotDeviceData_t    snapshotData;
} CSTASnapshotDeviceConfEvent_t;

typedef struct CSTASnapshotDeviceData_t {
    int    count; // count of calls on device
    struct    CSTASnapshotDeviceResponseInfo_t    *info;
                // info for each call
} CSTASnapshotDeviceData_t;

```

Syntax (Continued)

```
typedef struct CSTASnapshotDeviceResponseInfo_t {
    ConnectionID_t      callIdentifier;
                        // local connection ID
    CSTACallState_t    callstate;
                        // list of connection states
} CSTASnapshotDeviceResponseInfo_t;

typedef struct CSTACallState_t {
    int      count;      // count of connections on call
    LocalConnectionState_t *state;
                        // list of connection states
} CSTACallState_t;

typedef enum LocalConnectionState_t {
    CS_NONE      = -1,    // not an expected snapshot device
                        // response
    CS_NULL      = 0,    // indicates a bridged state
    CS_INITIATE  = 1,
    CS_ALERTING  = 2,
    CS_CONNECT   = 3,
    CS_HOLD      = 4,
    CS_QUEUED    = 5,
    CS_FAIL      = 6,
} LocalConnectionState_t;
```

Private Data Version 5 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTSnapshotDeviceConfEvent - Service Response Private
Data

typedef struct ATTEvent_t
{
    ATTEventType_t  eventType; // ATT_SNAPSHOT_DEVICE_CONF
    union
    {
        ATTSnapshotDeviceConfEvent_t  snapshotDevice;
    } u;
} ATTEvent_t;

typedef struct ATTSnapshotDeviceConfEvent_t
{
    int                count;
    ATTSnapshotDevice_t *pSnapshotDevice;
} ATTSnapshotDeviceConfEvent_t;

typedef struct ATTSnapshotDevice_t
{
    ConnectionID_t      call;
    ATTLocalCallState_t  state;
} ATTSnapshotDevice_t;

typedef enum ATTLocalCallState_t
{
    ATT_CS_INITIATED      = 1,
    ATT_CS_ALERTING      = 2,
    ATT_CS_CONNECTED     = 3,
    ATT_CS_HELD          = 4,
    ATT_CS_BRIDGED       = 5,
    ATT_CS_OTHER         = 6
} ATTLocalCallState_t;
```

Private Data Versions 2-4 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV4SnapshotDeviceConfEvent - Service Response Private
Data

typedef struct ATTEvent_t
{
    ATTEventType_t  eventType; // ATTV4_SNAPSHOT_DEVICE_CONF
    union
    {
        ATTV4SnapshotDeviceConfEvent_t  v4snapshotDevice;
    } u;
} ATTEvent_t;

typedef struct ATTV4SnapshotDeviceConfEvent_t
{
    int                count;
    ATTSnapshotDevice_t *pSnapshotDevice;
} ATTV4SnapshotDeviceConfEvent_t;

typedef struct ATTSnapshotDevice_t
{
    ConnectionID_t      call;
    ATTLocalCallState_t  state;
} ATTSnapshotDevice_t;

typedef enum ATTLocalCallState_t
{
    ATT_CS_INITIATED      = 1,
    ATT_CS_ALERTING      = 2,
    ATT_CS_CONNECTED     = 3,
    ATT_CS_HELD          = 4,
    ATT_CS_BRIDGED       = 5,
    ATT_CS_OTHER         = 6
} ATTLocalCallState_t;
```


Overview

There are three types of monitor services for the G3 PBX in Telephony Services. They all have the same confirmation event. The Change Monitor Filter Service is used by an application to change the filter options. The Monitor Stop Service is used to cancel a monitor service of any type. The Monitor Ended Event from the switch applies to any type of monitor services.

Change Monitor Filter Service — `cstaChangeMonitorFilter()`

This service is used by a client application to change the filter options in a previously requested monitor association.

Monitor Call Service — `cstaMonitorCall()`

This service provides call event reports passed by the call filter for a single call to an application, but does not provide any agent, feature, or maintenance event reports.

Monitor Calls Via Device Service — cstaMonitorCallsViaDevice

This service¹ provides call event reports passed by the call filter for all devices on all calls that involve a VDN or an ACD Split device. Event reports are provided for calls that arrive at the device after the monitor request is acknowledged. Events that occurred prior to the monitor request are not reported. If a call is diverted, forwarded, conferenced, or transferred to non-monitored ACD or VDN device, subsequent events of that call are reported. Special rules apply to the event reports when the call is diverted, forwarded, conferenced, or transferred. Details are provided in later sections.

This service does not provide any agent, feature, or maintenance event reports.

Monitor Device Service — cstaMonitorDevice()

This service² provides call event reports passed by the call filter for all devices on all calls at a station device. Event reports are provided for calls that occurred prior to the monitor request and arrive at the device after the monitor request is acknowledged. If a call is dropped, no further events of the call are reported, forwarded, or transferred from the device, and the device has ceased to participate in the call.

The service also provides feature event reports passed by the filter for a monitored station device as well as agent event reports passed by the filter for a monitored ACD Split device.

The service does not provide maintenance event reports.

Monitor Ended Event — CSTAMonitorEndedEvent

The switch uses this event report to notify a client application that a previously requested Monitor Service has been canceled.

Monitor Stop On Call Service (Private) — attMonitorStopOnCall()

An application uses this service to stop call event reports of a specific call on a monitored device.

-
1. The Monitor Calls Via Device Service is the call-type Monitor Start Service on a static device identifier in ECMA-179.
 2. The Monitor Device Service is the device-type Monitor Start Service on a static device identifier in ECMA-179.

Monitor Stop Service — cstaMonitorStop()

An application uses this service to cancel a previously requested Monitor Service.

Event Filters and Monitor Services

Table 8-1 shows the relation of event filters and monitor services. An “On” means that this filter is always turned on in the service request confirmation event or the change filter service request confirmation event. This monitor request will never receive this event.

An “On/Off” means that this filter can be turned on or off in the service request or in the change filter service request and the active filters will be specified in the confirmation event. If a filter is set to on, this monitor request will not receive that event.

If the Private Filter is set to On, all ATT private event filters (Entered Digits) will be automatically set to On, meaning that there will be no ATT private events for the monitor request.

Table 8-1. Event Filters and Monitor Services

Event Filters	Monitor Call	Monitor Device (Station)	Monitor Device (ACD Split)	Monitor Device (Trunk or All Trunks)	Monitor Calls Via Device (VDN or ACD Split)
Call Event Filters					
Advice of Charge (private data 5)	On/Off	On/Off	On/Off	On/Off	On/Off
Call Cleared	On/Off	On	On	On	On/Off
Conferenced	On/Off	On/Off	On	On	On/Off
Connection Cleared	On/Off	On/Off	On	On	On/Off
Delivered	On/Off	On/Off	On	On	On/Off
Diverted	On	On/Off	On	On	On/Off
Entered Digits (private)	On/Off	On	On	On	On/Off
Established	On/Off	On/Off	On	On	On/Off
Failed	On/Off	On/Off	On	On	On/Off
Held	On/Off	On/Off	On	On	On/Off

Issue 1 — December 2001

Table 8-1. Event Filters and Monitor Services

Event Filters	Monitor Call	Monitor Device (Station)	Monitor Device (ACD Split)	Monitor Device (Trunk or All Trunks)	Monitor Calls Via Device (VDN or ACD Split)
Network Reached	On/Off	On/Off	On	On	On/Off
Originated	On	On/Off ¹	On/Off ¹	On	On
Queued	On/Off	On/Off	On	On	On/Off
Retrieved	On/Off	On/Off	On	On	On/Off
Service Initiated	On	On/Off	On	On	On
Transferred	On/Off	On/Off	On	On	On/Off
Agent Event Filters					
Logged On	On	On/Off ¹	On/Off ¹	On	On
Logged Off	On	On/Off	On/Off	On	On
Not Ready	On	On	On	On	On
Ready	On	On	On	On	On
Work Not Ready	On	On	On	On	On
Work Ready	On	On	On	On	On
Feature Event Filters					
Call Information	On	On	On	On	On
Do Not Disturb	On	On	On	On	On
Forwarding	On	On	On	On	On
Message Waiting	On	On	On	On	On
Maintenance Event Filters					
Back in Service	On	On	On	On	On
Out of Service	On	On	On	On	On
Private Filter	On/Off	On/Off	On/Off	On/Off	On/Off

1. For PBX Version G3V3 and earlier, Originated and Agent Logged On are always filtered (On).

Local Connection Info and Monitor Services

Table 8-2 shows the availability of the `localConnectionInfo` parameter for the monitor services. These definitions follow the CSTA specification.

Table 8-2. Local Connection Information and Monitor Services

Parameter	Monitor Call	Monitor Device (Station)	Monitor Device (ACD Split)	Monitor Device (Trunk or All Trunks)	Monitor Calls Via Device (VDN or ACD Split)
<i>localConnectionInfo</i>	not supported	supported	not supported	not supported	not supported

Change Monitor Filter Service

Direction: Client to Switch

Function: *cstaChangeMonitorFilter()*

Confirmation Event: *CSTChangeMonitorFilterConfEvent*

Private Data Function: *attMonitorFilterExt()* (private data version 5),
attMonitorFilter() (private data versions 2-4)

Private Data Confirmation Event: *ATTMonitorConfEvent* (private data version 5), *ATTV4MontorConfEvent* (private data versions 2-4)

Service Parameters: *monitorCrossRefID, filterList*

Private Parameters: *privateFilter*

Ack Parameters: *filterList*

Ack Private Parameters: *usedFilter*

Nak Parameter: *universalFailure*

Functional Description:

The Change Monitor Filter Service is used by a client application to change the filter options in a previously requested monitor association.

Service Parameters:

- monitorCrossRefID*** [mandatory] Must be a valid Cross Reference ID that was returned in a previous CSTAMonitorConfEvent of this acsOpenStream session.
- filterList*** [mandatory — partially supported] Specifies the filters to be changed. Call Filter, Agent Filter, and Private Filter are supported.
- Setting a filter of an event (for example, CF_CALL_CLEARED=0x8000 is turned on) in the monitorFilter means that the event will be filtered out and no such event reports will be sent to the application.
- A zero Private Filter means that the application wants to receive the private events. If Private Filter is non-zero, private events will be filtered out. The Feature Filter and Maintenance Filter are not supported. If either is present, it will be ignored.

Private Parameters:

- privateFilter*** [optional] Specifies the G3 PBX private filters to be changed. The following G3 Private Call Filter and Call Event Reports are supported:
- Private data version 5:
 - ATT_ENTERED_DIGITS_FILTER
 - ATT_CHARGE_ADVICE_FILTER
 - Private data versions 2-4:
 - ATT_V4_ENTERED_DIGITS_FILTER
- See Table 8-1 to determine which filters are under the control of the application, that is, can be turned on and off.

Ack Parameters:

- filterList*** [optional — partially supported] Specifies the event reports that are to be filtered out on the object being monitored by the application. This may not be the filterList specified in the service request, because filters for events that are not supported by the G3 PBX and filters for events that do not apply to the monitored object are always turned on in filterList. All event reports in Maintenance Filter are set to ON, meaning that there are no reports supported for these events.

Ack Private Parameters:

usedFilter [optional] Specifies the G3 Private Event Reports that are to be filtered out on the object being monitored by the application.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a `CSTAUniversalFailureConfEvent`. The error parameter in this event may contain the following error values, or one of the error values described in the “`CSTAUniversalFailureConfEvent`” section in Chapter 3:

- `INVALID_CROSS_REF_ID` (17) The service request specified a Cross Reference ID that is not in use at this time.

Detailed Information:

See the “Event Report Detailed Information” section in Chapter 9, Event Report Service Group.

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaChangeMonitorFilter() - Service Request

RetCode_t    cstaChangeMonitorFilter (
    ACSHandle_t        acsHandle,
    InvokeID_t        invokeID,
    CSTAMonitorCrossRefID_t    monitorCrossRefID,
    CSTAMonitorFilter_t    *filterList,
    PrivateData_t        *privateData);

// CSTAChangeMonitorFilterConfEvent - Service Response

typedef struct
{
    ACSHandle_t        acsHandle;
    EventClass_t        eventClass;
                        // CSTACONFIRMATION
    EventType_t        eventType;
                        // CSTA_CHANGE_MONITOR_FILTER_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t    eventHeader;
    union
    {
        struct
        {
            InvokeID_t    invokeID;
            union
            {
                CSTAChangeMonitorFilterConfEvent_t
                changeMonitorFilter;
            } u;
        } cstaConfirmation;
    } event;
} CSTAEvent_t;

typedef struct CSTAChangeMonitorFilterConfEvent_t
{
    CSTAMonitorFilter_t    monitorFilter;
} CSTAChangeMonitorFilterConfEvent_t;

```

Syntax (Continued)

```
typedef unsigned short  CSTACallFilter_t;
#define                 CF_CALL_CLEARED      0x8000
#define                 CF_CONFERENCED      0x4000
#define                 CF_CONNECTION_CLEARED 0x2000
#define                 CF_DELIVERED        0x1000
#define                 CF_DIVERTED         0x0800
#define                 CF_ESTABLISHED      0x0400
#define                 CF_FAILED           0x0200
#define                 CF_HELD             0x0100
#define                 CF_NETWORK_REACHED  0x0080
#define                 CF_ORIGINATED       0x0040
#define                 CF_QUEUED           0x0020
#define                 CF_RETRIEVED        0x0010
#define                 CF_SERVICE_INITIATED 0x0008
#define                 CF_TRANSFERRED      0x0004

typedef unsigned char   CSTAFeatureFilter_t;
#define                 FF_CALL_INFORMATION 0x80
#define                 FF_DO_NOT_DISTURB   0x40
#define                 FF_FORWARDING       0x20
#define                 FF_MESSAGE_WAITING  0x10

typedef unsigned char   CSTAAgentFilter_t;

#define                 AF_LOGGED_ON         0x80
#define                 AF_LOGGED_OFF        0x40
#define                 AF_NOT_READY         0x20
#define                 AF_READY             0x10
#define                 AF_WORK_NOT_READY    0x08
// not supported
#define                 AF_WORK_READY        0x04

typedef unsigned char   CSTAMaintenanceFilter_t
// not supported
#define                 MF_BACK_IN_SERVICE   0x80
#define                 MF_OUT_OF_SERVICE    0x40

typedef struct CSTAMonitorFilter_t {
    CSTACallFilter_t call;
    CSTAFeatureFilter_t feature;
    CSTAAgentFilter_t agent;
    CSTAMaintenanceFilter_t maintenance; // not supported
    long privateFilter;
// 0 = private events
// non-zero = no private events
} CSTAMonitorFilter_t;
```

Private Data Version 5 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attMonitorFilterExt() - Service Request Private Data
// Setup Function

RetCode_t attMonitorFilterExt(
    ATTPrivateData_t *privateData,
    ATTPrivateFilter_t privateFilter);

typedef struct ATTPrivateData_t
{
    char vendor[32];
    ushort length;
    char data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef unsigned char ATTPrivateFilter_t;
#define ATT_ENTERED_DIGITS_FILTER 0x80
#define ATT_CHARGE_ADVICE_FILTER 0x40

// ATTMonitorConfEvent - Service Response Private Data

typedef struct ATTEvent_t
{
    ATTEventType_t eventType; // ATT_MONITOR_CONF
    union
    {
        ATTMonitorConfEvent_t monitorStart;
    } u;
} ATTEvent_t;

typedef struct ATTMonitorConfEvent_t
{
    ATTPrivateFilter_t usedFilter;
} ATTMonitorConfEvent_t;
```

Private Data Versions 2-4 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attMonitorFilter() - Service Request Private Data
// Setup Function

RetCode_t attMonitorFilterExt(
    ATTPrivateData_t *privateData,
    ATTV4PrivateFilter_t privateFilter);

typedef struct ATTPrivateData_t
{
    char vendor[32];
    ushort length;
    char data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef unsigned char ATTV4PrivateFilter_t;
#define ATTV4_ENTERED_DIGITS_FILTER 0x80

// ATTV4MonitorConfEvent - Service Response Private Data

typedef struct ATTEvent_t
{
    ATTEventType_t eventType; // ATTV4_MONITOR_CONF
    union
    {
        ATTV4MonitorConfEvent_t v4monitorStart;
    } u;
} ATTEvent_t;

typedef struct ATTV4MonitorConfEvent_t
{
    ATTV4PrivateFilter_t usedFilter;
} ATTV4MonitorConfEvent_t;
```

Monitor Call Service

Direction: Client to Switch

Function: *cstaMonitorCall()*

Confirmation Event: *CSTAMonitorConfEvent*

Private Data Function: *attMonitorFilterExt()* (private data version 5),
attMonitorFilter() (private data versions 2-4)

Private Data Confirmation Event: *ATTMonitorCallConfEvent* (private data version 5), *ATTV4MonitorCallConfEvent* (private data versions 2-4)

Service Parameters: *call, monitorFilter*

Private Parameters: *privateFilter*

Ack Parameters: *monitorCrossRefID, monitorFilter*

Ack Private Parameters: *usedFilter, snapshotCall*

Nak Parameter: *universalFailure*

Functional Description:

This service provides call event reports passed by the call filter for a call (call) already in progress. Event reports are provided after the monitor request is acknowledged. Events that occurred prior to the monitor request are not reported. A call that is being monitored may have a new call identifier assigned to it after a conference or transfer. In this case, event reports continue for that call with the new call identifier.

The event reports are provided for all endpoints directly connected to the G3 PBX and, in some cases, for endpoints not directly connected to the G3 PBX that are involved in a monitored call.

A snapshot of the call is provided in the *CSTAMonitorConfEvent*. The information provided is equivalent to the information provided in a *CSTASnapshotCallConfEvent* of the monitored call.

Only Call Filter/Call Event Reports and Private Filter are supported. Agent Event Reports, Feature Event Reports and Maintenance Event Reports are not provided.

Service Parameters:

- call*** [mandatory] ConnectionID of the call to be monitored.
- monitorFilter*** [optional — partially supported] Specifies the filters to be used with call. Only Call Filter/Call Event Reports and Private Filter are supported. If a Call Filter is not present, it defaults to no filter, meaning that all G3 PBX CSTA call events will be reported.
- Setting a filter of an event (for example, CF_CALL_CLEARED=0x8000 is turned on) in the monitorFilter means that the event will be filtered out and no such event reports will be sent to the application.
- A zero Private Filter means that the application wants to receive the private call events. If Private Filter is non-zero, private call events will be filtered out. The Agent Filter, Feature Filter, and Maintenance Filter are not supported. If one of these is present, it will be ignored.

Private Parameters:

- privateFilter*** [optional] Specifies the G3 PBX private filters to be changed. The following G3 Private Call Filter and Call Event Reports are supported:
- Private data version 5:
 - ATT_ENTERED_DIGITS_FILTER
 - ATT_CHARGE_ADVICE_FILTER
 - Private data versions 2-4:
 - ATT_V4_ENTERED_DIGITS_FILTER
- See Table 8-1 to determine which filters are under the control of the application, that is, can be turned on and off.

Ack Parameters:

monitorCrossRefID [mandatory] Contains the handle chosen by the G3 PBX Driver. This handle is a unique value within an acsOpenStream session for the duration of the monitor and is used by the application to correlate subsequent event reports to the monitor request that initiated them. It also allows the correlation of the Monitor Stop to the original cstaMonitorCall request.

monitorFilter [optional — partially supported] Specifies the event reports that are to be filtered out on the object being monitored by the application. This may not be the monitorFilter specified in the service request, because filters for events that are not supported by the G3 PBX and filters for events that do not apply to the monitored object are always turned on in monitorFilter. Only Call Filter and Call Event Reports are supported.

All event reports in Agent Filter, Feature Filter, Maintenance Filter, and Private Filter are set to ON, meaning that there are no reports supported for these events.

Ack Private Parameters:

usedFilter [optional] Specifies the G3 Private Filter and Event Reports that are to be filtered out on the object being monitored by the application.

snapshotCall [optional] Provides information about the device identifier, connection, and the CSTA Connection state for up to six (6) endpoints on the call. The Connection state may be one of the following: Unknown, Null, Initiated, Alerting, Queued, Connected, Held, or Failed. The information provided is equivalent to the information provided in a CSTASnapshotCallConfEvent of the monitored call.

Nak Parameters:

- universalFailure*** If the request is not successful, the application will receive a `CSTAUniversalFailureConfEvent`. The error parameter in this event may contain the following error values, or one of the error values described in the “`CSTAUniversalFailureConfEvent`” section in Chapter 3:
- `INVALID_CONNECTION_ID_FOR_ACTIVE_CALL` (23) (CS0/100) The call identifier is outside the range of the maximum call identifier value.
 - `NO_ACTIVE_CALL` (24) (CS3/86) The application has sent an invalid call identifier. Call does not exist or has been cleared.
 - `RESOURCE_BUSY` (33) G3PD is busy processing a `cstaMonitorCall` service request on the same call. Try again.
 - `GENERIC_SUBSCRIBED_RESOURCE_AVAILABILITY` (41) (CS0/50) The user has not subscribed for the requested service.
 - `OBJECT_MONITOR_LIMIT_EXCEEDED` (42) (CS3/40) The maximum number of calls being monitored on the G3 PBX was exceeded.
 - `OUTSTANDING_REQUEST_LIMIT_EXCEEDED` (44) (CS3/63) The same call may be monitored by another G3PD. The request cannot be executed because the system limit is exceeded for the maximum number of monitors on a call by G3PDs.

Detailed Information:

See also the section titled “Event Report Detailed Information” in Chapter 9, Event Report Service Group.

- **Monitor Ended Event Report** — When the monitored call is ended before a `cstaMonitorStop` is received to stop the `cstaMonitorCall` association, a `CSTAMonitorEndedEvent` will be sent to the application to terminate the `cstaMonitorCall` association.
- **Monitor Stop On Call Service** — When the `cstaMonitorCall` association is stopped by an `attMonitorStopOnCall` request before a `cstaMonitorStop` request is received, a `CSTAMonitorEndedEvent` will be sent to the application to terminate the `cstaMonitorCall` association.
- **Maximum Requests from Multiple G3PDs** — See the section titled “G3 CSTA System Capacity” in Chapter 3, G3 CSTA Services Overview.

- Multiple Application Requests — Multiple applications can have multiple `cstaMonitorCall` requests on one object through one G3PD. An application can have more than one `cstaMonitorCall` request on one object through one G3PD. However, this is not recommended.
- Advice of Charge Event Report (private data v5) — The `ATTChargeAdviceEvent` is provided, by an outside service, to streams which have enabled Advice of Charge using `attSetAdviceOfCharge()` and `cstaEscapeService()`. Typically, an `ATTChargeAdviceEvent` will arrive from the provider as a call ends, providing the final charge amount. Generally, the final `CSTAMonitorEndedEvent` (sent for call monitors at the end of a call) is delayed until that final `ATTChargeAdviceEvent` arrives. When there is a long delay in the arrival of the final `ATTChargeAdviceEvent`, the `CSTAMonitorEndedEvent` will be sent to the application and a final `ATTChargeAdviceEvent` will not be provided.

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaMonitorCall() - Service Request

RetCode_t    cstaMonitorCall (
    ACSHandle_t        acsHandle,
    InvokeID_t        invokeID,
    ConnectionID_t    *call,
    CSTAMonitorFilter_t    *monitorFilter, // supports
                                        // call filter only
    PrivateData_t    *privateData);

// CSTAMonitorConfEvent - Service Response

typedef struct
{
    ACSHandle_t        acsHandle;
    EventClass_t        eventClass;
    EventType_t        eventType;
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t    eventHeader;
    union
    {
        struct
        {
            InvokeID_t    invokeID;
            union
            {
                CSTAMonitorConfEvent_t    monitorStart;
            } u;
        } cstaConfirmation;
    } event;
} CSTAEvent_t;

typedef struct CSTAMonitorConfEvent_t
{
    CSTAMonitorCrossRefID_t    monitorCrossRefID;
    CSTAMonitorFilter_t        monitorFilter;
} CSTAMonitorConfEvent_t;

```

Syntax (Continued)

```

typedef long          CSTAMonitorCrossRefID_t;

typedef unsigned short CSTACallFilter_t;
#define              CF_CALL_CLEARED          0x8000
#define              CF_CONFERENCED          0x4000
#define              CF_CONNECTION_CLEARED   0x2000
#define              CF_DELIVERED           0x1000
#define              CF_DIVERTED            0x0800
#define              CF_ESTABLISHED         0x0400
#define              CF_FAILED              0x0200
#define              CF_HELD                0x0100
#define              CF_NETWORK_REACHED     0x0080
#define              CF_ORIGINATED          0x0040
#define              CF_QUEUED              0x0020
#define              CF_RETRIEVED           0x0010
#define              CF_SERVICE_INITIATED    0x0008
#define              CF_TRANSFERRED         0x0004

typedef unsigned char CSTAFeatureFilter_t;
// not supported
#define              FF_CALL_INFORMATION     0x80
#define              FF_DO_NOT_DISTURB      0x40
#define              FF_FORWARDING          0x20
#define              FF_MESSAGE_WAITING     0x10

typedef unsigned char CSTAAgentFilter_t;
#define              AF_LOGGED_ON            0x80
#define              AF_LOGGED_OFF          0x40
#define              AF_NOT_READY           0x20
#define              AF_READY               0x10
#define              AF_WORK_NOT_READY      0x08
#define              AF_WORK_READY          0x04

typedef unsigned char CSTAMaintenanceFilter_t;
// not supported
#define              MF_BACK_IN_SERVICE      0x80
#define              MF_OUT_OF_SERVICE       0x40

typedef struct CSTAMonitorFilter_t {
    CSTACallFilter_t      call;
    CSTAFeatureFilter_t   feature; // not supported
    CSTAMaintenanceFilter_t maintenance; // not supported
    long                  privateFilter;
// 0 = report private events
// non-zero = no private events
} CSTAAgentFilter_t agent;
CSTAMonitorFilter_t;

```

Private Data Version 5 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attMonitorFilterExt() - Service Request Private Data
// Setup Function

RetCode_t attMonitorFilterExt(
    ATTPrivateData_t *privateData,
    ATTPrivateFilter_t privateFilter);

typedef struct ATTPrivateData_t
{
    char vendor[32];
    ushort length;
    char data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef unsigned char ATTPrivateFilter_t;
#define ATT_ENTERED_DIGITS_FILTER 0x80
#define ATT_CHARGE_ADVICE_FILTER 0x40

// ATTMonitorCallConfEvent - Service Response Private Data

typedef struct ATTEvent_t
{
    ATTEventType_t eventType; // ATT_MONITOR_CALL_CONF
    union
    {
        ATTMonitorCallConfEvent_t monitorCallStart;
    } u;
} ATTEvent_t;

typedef struct ATTMonitorCallConfEvent_t
{
    ATTPrivateFilter_t usedFilter;
    ATTSnapshotCall_t snapshotCall;
} ATTMonitorCallConfEvent_t;

typedef struct ATTSnapshotCall_t
{
    int count;
    CSTASnapshotCallResponseInfo_t *pInfo;
} ATTSnapshotCall_t;

typedef struct CSTASnapshotCallResponseInfo_t
{
    SubjectDeviceID_t deviceOnCall;
    ConnectionID_t callIdentifier;
    LocalConnectionState_t localConnectionState;
} CSTASnapshotCallResponseInfoEvent_t;
```

Private Data Versions 2-4 Syntax

```

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attMonitorFilter() - Service Request Private Data
// Setup Function

RetCode_t attMonitorFilter(
    ATTPrivateData_t *privateData,
    ATTV4PrivateFilter_t privateFilter);

typedef struct ATTPrivateData_t
{
    char vendor[32];
    ushort length;
    char data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef unsigned char ATTV4PrivateFilter_t;
#define ATT_V4_ENTERED_DIGITS_FILTER 0x80

// ATTV4MonitorCallConfEvent - Service Response Private Data

typedef struct ATTEvent_t
{
    ATTEventType_t eventType; // ATTV4_MONITOR_CALL_CONF

    union
    {
        ATTV4MonitorCallConfEvent_t v4monitorCallStart;
    } u;
} ATTEvent_t;

typedef struct ATTV4MonitorCallConfEvent_t
{
    ATTV4PrivateFilter_t usedFilter;
    ATTV4SnapshotCall_t snapshotCall;
} ATTV4MonitorCallConfEvent_t;

typedef struct ATTV4SnapshotCall_t
{
    short count;
    CSTASnapshotCallResponseInfo_t info[ATT_MAX_PARTIES_ON_CALL];
} ATTV4SnapshotCall_t;

typedef struct CSTASnapshotCallResponseInfo_t
{
    SubjectDeviceID_t deviceOnCall;
    ConnectionID_t callIdentifier;
    LocalConnectionState_t localConnectionState;
} CSTASnapshotCallResponseInfoEvent_t;

```

Monitor Calls Via Device Service

Direction: Client to Switch

Function: *cstaMonitorCallsViaDevice()*

Confirmation Event: *CSTAMonitorConfEvent*

Private Data Function: *attMonitorFilterExt()* (private data version 5),
attMonitorFilter() (private data versions 2-4)

Private Data Confirmation Event: *ATTMonitorConfEvent* (private data version 5), *ATTV4MonitorConfEvent* (private data versions 2-4)

Service Parameters: *deviceID, monitorFilter*

Private Parameters: *privateFilter*

Ack Parameters: *monitorCrossRefID, monitorFilter*

Ack Private Parameters: *usedFilter*

Nak Parameter: *universalFailure*

Functional Description:

This service provides call event reports passed by the call filter for all devices on all calls that involve the device (deviceID). Event reports are provided for calls that arrive at the device after the monitor request is acknowledged. Events for calls that occurred prior to the monitor request are not reported. There are feature interactions between two *cstaMonitorCallsViaDevice* requests on different monitored ACD or VDN devices.³ If a call is diverted, forwarded, conferenced, or transferred to ACD or VDN device that is not monitored by another *cstaMonitorCallsViaDevice* request, subsequent call events are reported. If a call is diverted, forwarded, conferenced, or transferred to another device that is monitored by another *cstaMonitorCallsViaDevice* request, special rules apply to the call event reports. These rules are described in the "Detailed Information:" section.

The event reports are provided for all endpoints directly connected to the G3 PBX and may be present for certain types of endpoints not directly connected to the G3 PBX that are involved in the monitored device.

This service supports only VDN and ACD Split devices, but not station devices. Use *cstaMonitorDevice* service to monitor stations.

Only Call Filter/Call Event Reports and Private Filter are supported. Agent Event Reports, Feature Event Reports, and Maintenance Event Reports are not supported.

3. There are no feature interactions between a *cstaMonitorCallsViaDevice* request and a *cstaMonitorDevice* request. There are no feature interactions between a *cstaMonitorDevice* request and another *cstaMonitorDevice* request.

Service Parameters:

- deviceID*** [mandatory] A valid on-PBX VDN or ACD Split extension to be monitored. A station extension is invalid..
- monitorFilter*** [optional — partially supported] Specifies the filters to be used with deviceID. Only Call Filter/Call Event Reports and Private Filter are supported. If a Call Filter is not present, it defaults to no filter, meaning that all G3 PBX CSTA call events will be reported.
- Setting a filter of an event (for example, CF_CALL_CLEARED=0x8000 is turned on) in the monitorFilter means that the event will be filtered out and no such event reports will be sent to the application.
- A zero Private Filter means that the application wants to receive the private call events. If Private Filter is non-zero, private call events will be filtered out.
- The Agent Filter, Feature Filter, and Maintenance Filter are not supported. If one of these is present, it will be ignored.

Private Parameters:

- privateFilter*** [optional] Specifies the G3 PBX private filters to be changed. The following G3 Private Call Filter and Call Event Reports are supported:
- Private data version 5:
 - ATT_ENTERED_DIGITS_FILTER
 - ATT_CHARGE_ADVICE_FILTER
 - Private data versions 2-4:
 - ATT_V4_ENTERED_DIGITS_FILTER
- See Table 8-1 to determine which filters are under the control of the application, that is, can be turned on and off.

Ack Parameters:

monitorCrossRefID [mandatory] Contains the handle chosen by the G3 PBX Driver. This handle is a unique value within an acsOpenStream session for the duration of the monitor and is used by the application to correlate subsequent event reports to the monitor request that initiated them. It also allows the correlation of the Monitor Stop to the original cstaMonitorCallsViaDevice request.

monitorFilter [optional — partially supported] Specifies the event reports that are to be filtered out for the object being monitored by the application. This may not be the monitorFilter specified in the service request because filters for events that are not supported by the G3 PBX and filters for events that do not apply to the monitored device are always turned on in monitorFilter. Only Call Filter and Call Event Reports are supported.

All event reports in Agent Filter, Feature Filter, Maintenance Filter are set to "ON", meaning that there are no reports supported for these events.

Ack Private Parameter:

usedFilter [optional] Specifies the G3 private event reports that are to be filtered out on the object being monitored by the application.

Nak Parameters:

- universalFailure*** If the request is not successful, the application will receive a `CSTAUniversalFailureConfEvent`. The error parameter in this event may contain the following error values, or one of the error values described in the “`CSTAUniversalFailureConfEvent`” section in Chapter 3:
- `REQUEST_INCOMPATIBLE_WITH_OBJECT` (2) Monitored object is not administered correctly in the switch. The monitored object is an adjunct-controlled split or a vector-controlled split.
 - `INVALID_CSTA_DEVICE_IDENTIFIER` (12) An invalid device identifier or extension is specified in `deviceId`.
 - `RESOURCE_BUSY` (33) G3PD is busy processing a `cstaMonitorCallsViaDevice` service request on the same device. Try again.
 - `GENERIC_SUBSCRIBED_RESOURCE_AVAILABILITY` (41) The user has not subscribed to the requested service.
 - `OBJECT_MONITOR_LIMIT_EXCEEDED` (42) The request cannot be executed because the system limit would be exceeded for the maximum number of monitors.

Detailed Information:

See also the section titled “Event Report Detailed Information” in Chapter 9, Event Report Service Group.

- ACD split — An ACD split can be monitored by this service only for Call Event Reports.
- Adjunct-Controlled Splits — A `cstaMonitorCallsViaDevice` request will be denied (`REQUEST_INCOMPATIBLE_WITH_OBJECT`) if the monitored object is an adjunct- controlled split.
- Maximum Number of Objects that can be Monitored — See “G3 CSTA System Capacity” section in Chapter 3. G3 CSTA Services Overview.
- Multiple Requests — Multiple applications can have multiple `cstaMonitorCallsViaDevice` requests on one object. An application can have more than one `cstaMonitorCallsViaDevice` request on one object; however, the latter is not recommended.
- Personal Central Office Line (PCOL) — Members of a PCOL may be monitored. PCOL behaves like bridging for the purpose of event reporting.

- Skill Hunt Groups — A skill hunt group (split) cannot be monitored directly by an application. The VDN providing access to the vector(s) controlling the hunt group can be monitored instead, if event reports for calls delivered to the hunt group are desired.
- Special Rules — The following rules apply when a monitored call is diverted, forwarded, conferenced, or transferred.
 1. If a call monitored by a `cstaMonitorCallsViaDevice` request is diverted to a device that is not monitored by a `cstaMonitorCallsViaDevice` request, then there is no Diverted Event Report generated. Subsequent event reports of the call continue.
 2. If a call monitored by a `cstaMonitorCallsViaDevice` at an ACD or VDN device (A) and is diverted to an ACD or VDN device (B) monitored by a `cstaMonitorCallsViaDevice` request, then a Diverted Event Report is sent on the monitor for the device (A) that the call left, and no subsequent event reports will be sent for this call on the monitor for device (A). A Delivered Event Report is sent to the monitor for device (B) and subsequent call event reports are sent on the monitor for device (B). The rule is that call event reports of a call are sent to only one `cstaMonitorCallsViaDevice` request.
 3. If a call monitored by a `cstaMonitorCallsViaDevice` request is merged by a conference/transfer operation with a call that is monitored by a `cstaMonitorCallsViaDevice` request, a Call Ended Event Report is sent to the monitor request that the call left and no subsequent event reports are sent to this monitor request. A Conferenced/Transferred Event Report is sent to the monitor request with which the call stays and subsequent event reports of the call continue to be sent to this monitor request.
 4. If a call that is monitored by a `cstaMonitorCallsViaDevice` request is merged by a conference/transfer operation with a call that is not monitored by a `cstaMonitorCallsViaDevice` request and the resulting call is the one being monitored, a Conferenced/Transferred Event Report is sent to the monitor request and subsequent event reports of the call continue to the same monitor request. If the resulting call is the one not being monitored, a Conferenced/Transferred Event Report with a new callID is sent to the monitor request, a Call Ended Event Report is sent to the monitor request for the abandoned call, and subsequent event reports of the new call continue to be sent to the same request. In this case, the callID for the abandoned call is no longer valid.
- Station — A station cannot be monitored by this service.
- Terminating Extension Group (TEG) — Members of a TEG may be monitored. PCOL behaves like bridging for the purpose of event reporting.
- Vector-Controlled Split — A vector-controlled split cannot be monitored. The VDN providing access to the vector(s) controlling the split should be monitored instead.

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaMonitorCallsViaDevice() - Service Request

RetCode_t    cstaMonitorCallsViaDevice (
    ACSHandle_t        acsHandle,
    InvokeID_t        invokeID,
    DeviceID_t        *deviceID,
                    // must be VDN or ACD split
    CSTAMonitorFilter_t    *monitorFilter, // supports
                    // call filter only
    PrivateData_t        *privateData);

// CSTAMonitorConfEvent - Service Response

typedef struct
{
    ACSHandle_t        acsHandle;
    EventClass_t        eventClass;
                    // CSTACONFIRMATION
    EventType_t        eventType;
                    // CSTA_MONITOR_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t    eventHeader;
    union
    {
        struct
        {
            InvokeID_t    invokeID;
            union
            {
                CSTAMonitorConfEvent_t    monitorStart;
            } u;
        } cstaConfirmation;
    } event;
} CSTAEvent_t;

typedef struct CSTAMonitorConfEvent_t
{
    CSTAMonitorCrossRefID_t    monitorCrossRefID;
    CSTAMonitorFilter_t        monitorFilter;
} CSTAMonitorConfEvent_t;

```

Syntax (Continued)

```
typedef long          CSTAMonitorCrossRefID_t;

typedef unsigned short CSTACallFilter_t;
#define               CF_CONFERENCED           0x4000
#define               CF_CONNECTION_CLEARED    0x2000
#define               CF_DELIVERED           0x1000
#define               CF_DIVERTED           0x0800
#define               CF_ESTABLISHED         0x0400
#define               CF_FAILED             0x0200
#define               CF_HELD               0x0100
#define               CF_NETWORK_REACHED     0x0080
#define               CF_ORIGINATED         0x0040
#define               CF_QUEUED             0x0020
#define               CF_RETRIEVED         0x0010
#define               CF_SERVICE_INITIATED   0x0008
#define               CF_TRANSFERRED        0x0004

typedef unsigned char CSTAFeatureFilter_t;
// not supported
#define               FF_CALL_INFORMATION     0x80
#define               FF_DO_NOT_DISTURB     0x40
#define               FF_FORWARDING         0x20
#define               FF_MESSAGE_WAITING    0x10

typedef unsigned char CSTAAgentFilter_t;
// not supported
#define               AF_LOGGED_ON          0x80
#define               AF_LOGGED_OFF         0x40
#define               AF_NOT_READY          0x20
#define               AF_READY              0x10
#define               AF_WORK_NOT_READY     0x08
#define               AF_WORK_READY         0x04

typedef unsigned char CSTAMaintenanceFilter_t;
// not supported
#define               MF_BACK_IN_SERVICE     0x80
#define               MF_OUT_OF_SERVICE     0x40

typedef struct CSTAMonitorFilter_t {
    CSTACallFilter_t    call;
    CSTAFeatureFilter_t feature; // not supported
    CSTAMaintenanceFilter_t maintenance; // not supported
    long                privateFilter;
                        // 0 = report private events
                        // non-zero = no private events
} CSTAMonitorFilter_t;
```

Private Data Version 5 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attMonitorFilterExt() - Service Request Private Data
// Setup Function

RetCode_t attMonitorFilterExt(
    ATTPrivateData_t *privateData,
    ATTPrivateFilter_t privateFilter);

typedef struct ATTPrivateData_t
{
    char vendor[32];
    ushort length;
    char data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef unsigned char ATTPrivateFilter_t;
#define ATT_ENTERED_DIGITS_FILTER 0x80
#define ATT_CHARGE_ADVICE_FILTER 0x40

// ATTMonitorConfEvent - Service Response Private Data

typedef struct ATTEvent_t
{
    ATTEventType_t eventType; // ATT_MONITOR_CONF
    union
    {
        ATTMonitorConfEvent_t monitorStart;
    } u;
} ATTEvent_t;

typedef struct ATTMonitorConfEvent_t
{
    ATTPrivateFilter_t usedFilter;
} ATTMonitorConfEvent_t;
```

Private Data Versions 2-4 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attMonitorFilterExt() - Service Request Private Data
// Setup Function

RetCode_t attMonitorFilterExt(
    ATTPrivateData_t      *privateData,
    ATTV4PrivateFilter_t  privateFilter);

typedef struct ATTPrivateData_t
{
    char      vendor[32];
    ushort   length;
    char      data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef unsigned char  ATTV4PrivateFilter_t;
#define                ATT_V4_ENTERED_DIGITS_FILTER 0x80

// ATTV4MonitorConfEvent - Service Response Private Data

typedef struct ATTEvent_t
{
    ATTEventType_t  eventType;      // ATTV4_MONITOR_CONF
    union
    {
        ATTV4MonitorConfEvent_t  monitorStart;
    } u;
} ATTEvent_t;

typedef struct ATTV4MonitorConfEvent_t
{
    ATTV4PrivateFilter_t  usedFilter;
} ATTV4MonitorConfEvent_t;
```

Monitor Device Service

Direction: Client to Switch

Function: *cstaMonitorDevice()*

Confirmation Event: *CSTAMonitorConfEvent*

Private Data Function: *attMonitorFilterExt()* (private data version 5),
attMonitorFilter() (private data versions 2-4)

Private Data Confirmation Event: *ATTMonitorConfEvent* (private data version 5), *ATTV4MonitorConfEvent* (private data versions 2-4)

Service Parameters: *deviceId*, *monitorFilter*

Private Parameters: *privateFilter*

Ack Parameters: *monitorCrossRefID*, *monitorFilter*

Ack Private Parameters: *usedFilter*

Nak Parameter: *universalFailure*

Functional Description:

This service provides call event reports passed by the call filter for all devices on all calls at a device. Event reports are provided for calls that occurred previous to the monitor request and arrive at the device after the monitor request is acknowledged. Call events are also provided for calls already present at the device. No further events for a call are reported when that call is dropped, forwarded, or transferred, conferenced, or the device ceases to participate in the call.

The Call Cleared Event is never provided for this service. There are no subsequent event reports for a call after a Connection Cleared or a Diverted Event Report has been received for that call on this service. Reporting of the subsequent call event reports after a Transferred Event Report is dependent on whether the call is merged-in or merged-out from the monitored device.

The event reports are provided for all endpoints directly connected to the G3 PBX and may in certain cases be provided for endpoints not directly connected to the G3 PBX that are involved in the calls with the monitored device.

This service supports Call Event Reports for station devices as well as Agent Event Reports for ACD Split devices.

Maintenance Event Reports are not supported.

NOTE:

DEFINITY ECS Release 5 and later software supports the Charge Advice Event feature. To receive Charge Advice Events, an application must first turn the Charge Advice Event feature on using the Set Advice of Charge Service. (For details, see “Set Advice of Charge Service (Private Data Version 5 and Later)” in Chapter 5.) If the Charge Advice Event feature is turned on, a trunk group monitored by a *cstaMonitorDevice*, a station monitored by a *cstaMonitorDevice*, or a call monitored by a *cstaMonitorCall* will receive Charge Advice Events. However, this will not occur if the Charge Advice Event is filtered out by the *privateFilter* in the monitor request and its confirmation event.

Service Parameters:

deviceID

[mandatory] A valid on-PBX extension, trunk group, or ACD extension to be monitored. A VDN extension is invalid.

A trunk group number has the format of a 'T' followed by the trunk group number (e.g., T123), or a 'T' followed by a '#' to indicate all trunk groups (i.e., "T#").

- If a single trunk group number is specified, the monitor session will receive the Charge Advice Event for that trunk group only.
- If "T#" is specified, the monitor session will receive Charge Advice Events from all trunk groups.

A trunk group monitoring will receive the Charge Advice Event only. It will not receive any other call events.

monitorFilter

[optional — partially supported] Specifies the filters to be used with deviceID. Call Filter/Event Reports are supported for station device. If a Call Filter is not present, it defaults to no filter, meaning that all G3 CSTA Call Event Reports will be reported.

The Agent Filter is supported for ACD Split devices.

Setting a filter of an event (for example, CF_CALL_CLEARED=0x8000 is turned on) in the monitorFilter means that the event will be filtered out and no such event reports will be sent to the application.

A zero Private Filter means that the application wants to receive the private events. If Private Filter is non-zero, private events will be filtered out.

The Feature Filter and Maintenance Filter are not supported. If a filter that does not apply to the monitored device is present, it will be ignored.

Private Parameters:

privateFilter [optional] Specifies the G3 PBX private filters to be changed. The following G3 Private Call Filter and Call Event Reports are supported:

- Private data version 5:
 - ATT_ENTERED_DIGITS_FILTER
 - ATT_CHARGE_ADVICE_FILTER
- Private data versions 2-4:
 - ATT_V4_ENTERED_DIGITS_FILTER

See Table 8-1 to determine which filters are under the control of the application, that is, can be turned on and off.

Ack Parameters:

monitorCrossRefID [mandatory] Contains the handle chosen by the G3 PBX Driver. This handle is a unique value within an `acsOpenStream` session for the duration of the monitor and is used by the application to correlate subsequent event reports to the monitor request that initiated them. It also allows the correlation of the Monitor Stop to the original Monitor Service request.

monitorFilter [optional — partially supported] Specifies the event reports that are to be filtered out for the object being monitored by the application. This may not be the `monitorFilter` specified in the service request because filters for events that are not supported by the G3 PBX and filters for events that do not apply to the monitored device are always turned on in `monitorFilter`. Maintenance Filters are set to "ON", meaning that there are no reports supported for these events.

Ack Private Parameter:

usedFilter [optional] Specifies the G3 private event reports that are to be filtered out on the object being monitored by the application.

Nak Parameters:

- universalFailure*** If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:
- **INVALID_CSTA_DEVICE_IDENTIFIER (12)** An invalid device identifier or extension is specified in deviceID.
 - **RESOURCE_BUSY (33)** G3PD is busy processing a cstaMonitorDevice service request on the same device. Try again.
 - **GENERIC_SUBSCRIBED_RESOURCE_AVAILABILITY (41)** The user has not subscribed to the requested service. The Domain (Station) Control feature may not be turned on in the G3 PBX.
 - **OBJECT_MONITOR_LIMIT_EXCEEDED (42)** The request cannot be executed because the system limit would be exceeded for the maximum number of monitor.

Detailed Information:

See also the section titled “Event Report Detailed Information” in Chapter 9, Event Report Service Group.

- **ACD split** — An ACD split can be monitored by this service only for Agent Event Reports.
- **Administration Without Hardware (AWOH)** — A station administered without hardware may be monitored. However, no event reports will be provided to the application for this station since there will be no activity at such an extension.
- **Analog ports** — Analog ports equipped with modems can be monitored by the cstaMonitorDevice Service.
- **Attendants and Attendant Groups** — An attendant group extension or an individual attendant extension number cannot have a Monitor Device Service.
- **Feature Access Monitoring** — A station will not prohibit users from access to any enabled switch features. A monitored station can access any enabled switch feature.
- **Logical Agents** — A logical agent’s station extension can be monitored. Login IDs are not valid monitor objects.
- **Multiple Requests** — Multiple applications can have multiple cstaMonitorDevice requests on one object. An application can have more than one cstaMonitorDevice request on one object. However, this is not recommended.

- Personal Central Office Line (PCOL) — Members of a PCOL may be monitored. PCOL behaves like bridging for the purpose of event reporting.
- Skill Hunt Groups — A skill hunt group (split) cannot be monitored directly by an application. The VDN providing access to the vector(s) controlling the hunt group can be monitored instead if event reports for calls delivered to the hunt group are desired.
- Terminating Extension Group (TEG) — Members of a TEG may be monitored. PCOL behaves like bridging for the purpose of event reporting.
- VDN — A VDN cannot be monitored by this service.
- Vector-Controlled Split — A vector-controlled split cannot be monitored. The VDN providing access to the vector(s) controlling the split should be monitored instead.

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaMonitorDevice() - Service Request

RetCode_t    cstaMonitorDevice (
    ACSHandle_t        acsHandle,
    InvokeID_t        invokeID,
    DeviceID_t        *deviceID,
    CSTAMonitorFilter_t    *monitorFilter,
    PrivateData_t        *privateData);

typedef struct
{
    ACSHandle_t        acsHandle;
    EventClass_t        eventClass;
    EventType_t        eventType;
} ACSEventHeader_t;

// CSTAMonitorConf - Event

typedef struct
{
    ACSEventHeader_t    eventHeader;
    union
    {
        struct
        {
            InvokeID_t    invokeID;
            union
            {
                CSTAMonitorConfEvent_t    monitorStart;
            } u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAMonitorConfEvent_t
{
    CSTAMonitorCrossRefID_t    monitorCrossRefID;
    CSTAMonitorFilter_t        monitorFilter;
} CSTAMonitorConfEvent_t;

```

Syntax (Continued)

```
typedef long          CSTAMonitorCrossRefID_t;

typedef unsigned short CSTACallFilter_t;
#define              CF_CALL_CLEARED          0x8000
#define              CF_CONFERENCED          0x4000
#define              CF_CONNECTION_CLEARED    0x2000
#define              CF_DELIVERED            0x1000
#define              CF_DIVERTED             0x0800
#define              CF_ESTABLISHED           0x0400
#define              CF_FAILED                0x0200
#define              CF_HELD                  0x0100
#define              CF_NETWORK_REACHED      0x0080
#define              CF_ORIGINATED           0x0040
#define              CF_QUEUED                0x0020
#define              CF_RETRIEVED            0x0010
#define              CF_SERVICE_INITIATED     0x0008
#define              CF_TRANSFERRED          0x0004

typedef unsigned char CSTAFeatureFilter_t;
#define              FF_CALL_INFORMATION     0x80
#define              FF_DO_NOT_DISTURB      0x40
#define              FF_FORWARDING          0x20
#define              FF_MESSAGE_WAITING     0x10

typedef unsigned char CSTAAgentFilter_t;
#define              AF_LOGGED_ON            0x80
#define              AF_LOGGED_OFF           0x40
#define              AF_NOT_READY            0x20
#define              AF_READY                0x08
#define              AF_WORK_READY           0x04

typedef unsigned char CSTAMaintenanceFilter_t;
#define              MF_BACK_IN_SERVICE      0x80
#define              MF_OUT_OF_SERVICE       0x40

typedef struct CSTAMonitorFilter_t {
    CSTACallFilter_t      call;
    CSTAFeatureFilter_t   feature;
    CSTAMaintenanceFilter_t maintenance;
    long                  privateFilter;
                        // 0 = report private events
                        // non-zero = no private events
} CSTAMonitorFilter_t;
```

Private Data Version 5 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attMonitorFilterExt() - Service Request Private Data
// Setup Function

RetCode_t attMonitorFilterExt(
    ATTPrivateData_t *privateData,
    ATTPrivateFilter_t privateFilter);

typedef struct ATTPrivateData_t
{
    char vendor[32];
    ushort length;
    char data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef unsigned char ATTPrivateFilter_t;
#define ATT_ENTERED_DIGITS_FILTER 0x80
#define ATT_CHARGE_ADVICE_FILTER 0x40

// ATTMonitorConfEvent - Service Response Private Data

typedef struct ATTEvent_t
{
    ATTEventType_t eventType; // ATT_MONITOR_CONF
    union
    {
        ATTMonitorConfEvent_t monitorStart;
    } u;
} ATTEvent_t;

typedef struct ATTMonitorConfEvent_t
{
    ATTPrivateFilter_t usedFilter;
} ATTMonitorConfEvent_t;
```

Private Data Versions 2-4 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attMonitorFilter() - Service Request Private Data
// Setup Function

RetCode_t attMonitorFilter(
    ATTPrivateData_t *privateData,
    ATTV4PrivateFilter_t privateFilter);

typedef struct ATTPrivateData_t
{
    char vendor[32];
    ushort length;
    char data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef unsigned char ATTV4PrivateFilter_t;
#define ATT_V4_ENTERED_DIGITS_FILTER 0x80

// ATTV4MonitorConfEvent - Service Response Private Data

typedef struct ATTEvent_t
{
    ATTEventType_t eventType; // ATTV4_MONITOR_CONF
    union
    {
        ATTV4MonitorConfEvent_t v4monitorStart;
    } u;
} ATTEvent_t;

typedef struct ATTV4MonitorConfEvent_t
{
    ATTV4PrivateFilter_t usedFilter;
} ATTV4MonitorConfEvent_t;
```

Monitor Ended Event Report

Direction: Switch to Client
Event: *CSTAMonitorEndedEvent*
Service Parameters: *monitorCrossRefID*

Functional Description:

The G3 PBX uses the Monitor Ended Event Report to cancel a subscription to a previously requested *cstaMonitorCall*, *cstaMonitorDevice* or *cstaMonitorCallsViaDevice* Service when a monitor object is removed or changed to become an invalid object by switch administration or when the switch can no longer provide the information. Once a Monitor Ended Event Report is generated, event reports cease to be sent to the client application by the switch and the Cross Reference Association that was established by the original service request is terminated.

Service Parameters:

- | | |
|--------------------------|---|
| <i>monitorCrossRefID</i> | [mandatory] Must be a valid Cross Reference ID of this <i>acsOpenStream</i> session. |
| <i>cause</i> | [optional — supported] Specifies the reason for this event.
The following Event Causes are explicitly sent from the switch: <ul style="list-style-type: none">■ <i>EC_NETWORK_NOT_OBTAINABLE</i> The previously monitored object is no longer available due to a CTI link failure.■ <i>EC_RESOURCES_NOT_AVAILABLE</i> The previously monitored object is no longer available or valid due to switch administration changes or communication protocol error. |

Detailed Information:

See the section titled “Event Report Detailed Information” in Chapter 9, Event Report Services Group.

Syntax

```

#include <acs.h>
#include <csta.h>

typedef struct
{
    ACSHandle_t          acsHandle;
    EventClass_t        eventClass;
                        // CSTAUNSOLICITED
    EventType_t         eventType;
                        // CSTA_MONITOR_ENDED
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t    eventHeader;
    union
    {
        struct
        {
            CSTAMonitorCrossRefID_t monitorCrossRefID;
            union
            {
                CSTAMonitorEndedEvent_t monitorEnded;
            } u;
        } cstaUnsolicited;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAMonitorEndedEvent_t
{
    CSTAEventCause_t    cause;
} CSTAMonitorEndedEvent_t;

```

Monitor Stop On Call Service (Private)

Direction: Client to Switch

Function: *cstaEscapeService()*

Confirmation Event: *CSTAEscapeServiceConfEvent*

Private Data Function: *attMonitorStopOnCall()*

Private Data Confirmation Event: *ATTMonitorStopOnCallConfEvent*

Private Parameters: *monitorCrossRefID, callID*

Ack Parameters: *noData*

Ack Private Parameters: *noData*

Nak Parameter: *universalFailure*

Functional Description:

An application uses the Monitor Stop On Call Service to stop Call Event Reports of a specific call reported by a *cstaMonitorCall*, *cstaMonitorDevice* or *cstaMonitorCallsViaDevice* Service when it no longer has an interest in that call. Once a Monitor Stop On Call request has been acknowledged, event reports of that call cease to be sent to the client application. The Monitor Cross Reference Association that was established by the original *cstaMonitorDevice* or *cstaMonitorCallsViaDevice* Service request continues.

If this service applies to a *cstaMonitorCall* association, the association will be terminated by a Monitor Ended Event Report.

⇒ NOTE:

The current release provides this capability for monitors initiated with the *cstaMonitorCall* service only. It does not work for the other types of monitors.

Private Parameters:

monitorCrossRefID [mandatory] Must be a valid Cross Reference ID that was returned in a previous *CSTAMonitorConfEvent* of this *acsOpenStream* session.

callID [mandatory] This is the *callID* of the call whose event reports are to be stopped.

Ack Parameters:

noData None for this service.

Ack Private Parameters:

noData None for this service.

Nak Parameters:

- universalFailure*** If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:
- INVALID_CROSS_REF_ID (17) The service request specified a Cross Reference ID that is not in use at this time.
 - NO_ACTIVE_CALL (24) The application has sent an invalid call identifier. The call does not exist, the call has been cleared, or the call is not being monitored by the monitoring device.

Detailed Information:

See also the section titled “Event Report Detailed Information” in Chapter 9, Event Report Services Group.

- This service will take effect immediately. Event reports to the application for the specified call will cease after this monitor request. The switch continues to process the call at the monitored object. Call processing is not affected by this service.
- This service will not affect Call Event Reports of the specified call on other monitoring associations.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaEscapeService() - Service Request

RetCode_t    cstaEscapeService (
    ACSHandle_t        acsHandle,
    InvokeID_t        invokeID,
    PrivateData_t     *privateData);

// CSTAEscapeServiceConfEvent - Service Response

typedef struct
{
    ACSHandle_t        acsHandle;
    EventClass_t       eventClass;
    // CSTACONFIRMATION
    EventType_t       eventType;
    // CSTA_ESCAPE_SERVICE_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t   eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
            union
            {
                CSTAEscapeSvcConfEvent_t  escapeService;
            } u;
        } cstaConfirmation;
    } event;
} CSTAEvent_t;

typedef struct CSTAEscapeSvcConfEvent_t
{
    Nulltype           null;
} CSTAEscapeSvcConfEvent_t;
```

Private Parameter Syntax

```

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attMonitorStopOnCall() - Service Request Private Data
// Setup Function

RetCode_t    attMonitorStopOnCall(
    ATTPrivateData_t    *privateData,
    CSTAMonitorCrossRefID_t    monitorCrossRefID,
    ConnectionID_t    *call);

// ATTMonitorStopOnCallEvent - Service Response Private Data

```

⇒ NOTE:

If private data accompanies CSTAMonitorStopOnCallConfEvent, then the private data would be stored in the location that the application specified as the privateData parameter in the acsGetEventBlock() or acsGetEventPoll() request. If the privateData pointer is set to NULL in these requests, then CSTAMonitorStopOnCallConfEvent does not deliver private data to the application. If the acsGetEventBlock() or acsGetEventPoll() returns Private Data length of 0, then no private data is provided with this event.

```

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATT private data event structure:

typedef struct
{
    ATTEventType    eventType;
                    // ATT_MONITOR_STOP_ON_CALL_CONF

    union
    {
        ATTMonitorStopOnCallConfEvent_t    monitorStop;
    }u;
} ATTEvent_t;

typedef struct ATTMonitorStopOnCallConfEvent_t {
    Nulltype    null;
} ATTMonitorStopOnCallConfEvent_t;

```

Monitor Stop Service

Direction: Client to Switch
Function: *cstaMonitorStop()*
Confirmation Event: *CSTAMonitorStopConfEvent*
Service Parameters: *monitorCrossRefID*
Ack Parameters: *noData*
Nak Parameter: *universalFailure*

Functional Description:

An application uses the Monitor Stop Service to cancel a subscription to a previously requested *cstaMonitorCall*, *cstaMonitorDevice*, or *cstaMonitorCallsViaDevice* Service when it no longer has an interest in continuing a monitor. Once a Monitor Stop request has been acknowledged, event reports cease to be sent to the client application by the switch and the Cross Reference Association that was established by the original service request is terminated.

Private Parameter:

monitorCrossRefID [mandatory] Must be a valid Cross Reference ID that was returned in a previous *CSTAMonitorConfEvent* of this *acsOpenStream* session.

Ack Parameter:

noData None for this service.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a *CSTAUniversalFailureConfEvent*. The error parameter in this event may contain the following error values, or one of the error values described in the “*CSTAUniversalFailureConfEvent*” section in Chapter 3:

- *INVALID_CROSS_REF_ID* (17) The service request specified a Cross Reference ID that is not in use at this time.

Detailed Information:

See also the “Event Report Detailed Information” section in Chapter 9, Event Report Services Group.

- **Switch Operation** — This service will take effect immediately. Event reports to the application for calls in progress will stop for this monitor request. The switch continues to process calls at the monitored object. Calls present at the monitored object are not affected by this service.

Syntax

```
#include <acs.h>
#include <csta.h>

RetCode_t    cstaMonitorStop (
    ACSHandle_t      acsHandle,
    InvokeID_t       invokeID,
    CSTAMonitorCrossRefID_t  monitorCrossRefID,
    PrivateData_t    *privateData);

typedef struct
{
    ACSHandle_t      acsHandle;
    EventClass_t     eventClass;
    EventType_t      eventType;
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t  eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
            union
            {
                CSTAMonitorStopConfEvent_t  monitorStop;
            } u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAMonitorStopConfEvent_t
{
    Nulltype          null;
} CSTAMonitorStopConfEvent_t;
```


CSTAEventCause and LocalConnectionState

Following are the definitions of the enumerated types CSTAEventCause and LocalConnectionState. These data structures are used extensively by the Event Report Service Group members described in this chapter.

```
typedef enum CSTAEventCause_t {
    EC_NONE = -1, // no cause value is specified
    EC_ACTIVE_MONITOR = 1,
    EC_ALTERNATE = 2,
    EC_BUSY = 3,
    EC_CALL_BACK = 4,
    EC_CALL_CANCELLED = 5,
    EC_CALL_FORWARD_ALWAYS = 6,
    EC_CALL_FORWARD_BUSY = 7,
    EC_CALL_FORWARD_NO_ANSWER = 8,
    EC_CALL_FORWARD = 9,
    EC_CALL_NOT_ANSWERED = 10,
    EC_CALL_PICKUP = 11,
    EC_CAMP_ON = 12,
    EC_DEST_NOT_OBTAINABLE = 13,
    EC_DO_NOT_DISTURB = 14,
    EC_INCOMPATIBLE_DESTINATION = 15,
    EC_INVALID_ACCOUNT_CODE = 16,
    EC_KEY_CONFERENCE = 17,
    EC_LOCKOUT = 18,
    EC_MAINTENANCE = 19,
    EC_NETWORK_CONGESTION = 20,
    EC_NETWORK_NOT_OBTAINABLE = 21,
```

```
    EC_NEW_CALL = 22,  
    EC_NO_AVAILABLE_AGENTS = 23,  
    EC_OVERRIDE = 24,  
    EC_PARK = 25,  
    EC_OVERFLOW = 26,  
    EC_RECALL = 27,  
    EC_REDIRECTED = 28,  
    EC_REORDER_TONE = 29,  
    EC_RESOURCES_NOT_AVAILABLE = 30,  
    EC_SILENT_MONITOR = 31,  
    EC_TRANSFER = 32,  
    EC_TRUNKS_BUSY = 33,  
    EC_VOICE_UNIT_INITIATOR = 34  
} CSTAEventCause_t;  
  
typedef enum LocalConnectionState_t {  
    CS_NONE = -1,    // state not known  
    CS_NULL = 0,  
    CS_INITIATE = 1,  
    CS_ALERTING = 2,  
    CS_CONNECT = 3,  
    CS_HOLD = 4,  
    CS_QUEUED = 5,  
    CS_FAIL = 6  
} LocalConnectionState_t;
```

Event Minimization Feature on G3 PBX

If Event Minimization is set to “y” for the CTI link connected to the G3PD, then only one set of events for a call is sent to the G3PD whether one or more devices are monitored. For example, if a VDN and an agent station are both monitored, only the VDN monitoring will received the Delivered Event.

⇒ NOTE:

The Event Minimization feature must be set to “n” on the switch for the CTI link connected to the G3PD.

Call Cleared Event

Direction: Switch to Client

Event: *CSTACallClearedEvent*

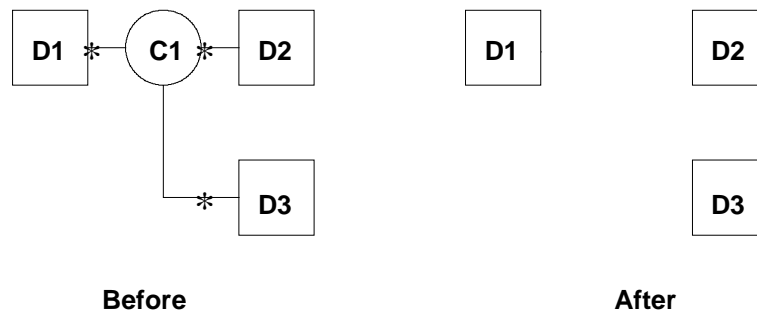
Private Data Event: *ATTCallClearedEvent*

Service Parameters: *monitorCrossRefID, clearedCall, localConnectionInfo, cause*

Private Parameters: *reason*

Functional Description:

The Call Cleared Event Report indicates that a call is ended. Normally this occurs when the last remaining device or party disconnects from the call. It can also occur when a call is immediately dissolved as the call being conferenced or transferred for a *cstaMonitorCallsViaDevice* request, but not for a *cstaMonitorDevice* request.



Service Parameters:

- monitorCrossRefID*** [mandatory] Contains the handle to the monitor request for which this event is reported.
- clearedCall*** [mandatory] Specifies the callID of the call that has been cleared. The deviceID is set to 0.
- localConnectionInfo*** [optional — supported] Always specifies a null state (CS_NULL).
- cause*** [optional — supported] Specifies a cause when the call is not terminated normally. EC_NONE is specified for normal call termination.
- EC_BUSY — Device busy.
 - EC_CALL_CANCELLED — Call rejected or canceled.
 - EC_DEST_NOT_OBTAINABLE — Called device is not reachable or wrong number is called.
 - EC_CALL_NOT_ANSWERED — Called device not responding or call not answered (maxRings timed out) for a MakePredictiveCall.
 - EC_NETWORK_CONGESTION — Network congestion or channel is unacceptable.
 - EC_RESOURCES_NOT_AVAILABLE — No circuit or channel is available.
 - EC_TRANSFER — Call merged due to transfer or conference.
 - EC_REORDER_TONE — Intercept SIT treatment - Number changed.
 - EC_VOICE_UNIT_INITIATOR — Answer machine is detected for a MakePredictiveCall.

Private Parameters:

- reason*** [optional] Specifies the reason for this event. The following reason codes are supported:
- AR_NONE— indicate no value specified for reason.
 - AR_ANSWER_NORMAL — Answer supervision from the network or internal answer.
 - AR_ANSWER_TIMED — Assumed answer based on internal timer.
 - AR_ANSWER_VOICE_ENERGY — Voice energy detection from a call classifier.

- AR_ANSWER_MACHINE_DETECTED — Answering machine detected
- AR_SIT_REORDER — Switch equipment congestion
- AR_SIT_NO_CIRCUIT — No circuit or channel available
- AR_SIT_INTERCEPT — Number changed
- AR_SIT_VACANT_CODE — Unassigned number
- AR_SIT_INEFFECTIVE_OTHER — Invalid number
- AR_SIT_UNKNOWN — Normal unspecified

Detailed Information:

See the “Event Report Detailed Information” section in this chapter.

Syntax

```
#include <acs.h>
#include <csta.h>

// CSTACallClearedEvent

typedef struct
{
    ACSHandle_t      acsHandle;
    EventClass_t    eventClass;    // CSTAUNSOLICITED
    EventType_t     eventType;    // CSTA_CALL_CLEARED
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            CSTAMonitorCrossRefID_t monitorCrossRefId;
            union
            {
                CSTACallClearedEvent_t callCleared;
            } u;
        } cstaUnsolicited;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTACallClearedEvent_t
{
    ConnectionID_t   clearedCall;
                    // DeviceID is always 0
    LocalConnectionState_t localConnectionInfo;
                    // always CS_NULL
    CSTAEventCause_t cause;
} CSTACallClearedEvent;
```

Private Parameter Syntax

If private data accompanies a CSTACallClearedEvent, then the private data would be stored in the location that the application specified as the private data parameter in the acsGetEventBlock() or acsGetEventPoll() request. If the privateData pointer is set to NULL in these requests, then the CSTACallClearedEvent does not deliver private data to the application. If the acsGetEventBlock() or acsGetEventPoll() returns a Private Data length of 0, then no private data is provided with this event.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTCallClearedEvent - CSTA Unsolicited Event Private Data

typedef struct ATTEvent_t
{
    ATTEventType_teventType;// ATT_CALL_CLEARED
    union
    {
        ATTCallClearedEvent_tcallClearedEvent;
    } u;
} ATTEvent_t;

typedef struct ATTCallClearedEvent_t
{
    ATReasonCode_treason;
} ATTCallClearedEvent_t;

typedef enum ATReasonCode_t {
AR_NONE          = 0, // no reason code specified
AR_ANSWER_NORMAL = 1, // answer supervision from
                    // the network or internal
                    // answer
AR_ANSWER_TIMED  = 2, // assumed answer based on
                    // internal timer
AR_ANSWER_VOICE_ENERGY= 3, // voice energy detection by
                    // classifier
AR_ANSWER_MACHINE_DETECTED = 4, // answering machine detected
AR_SIT_REORDER   = 5, // switch equipment congestion
AR_SIT_NO_CIRCUIT = 6, // no circuit or channel available
AR_SIT_INTERCEPT = 7, // number changed
AR_SIT_VACANT_CODE= 8, // unassigned number
AR_SIT_INEFFECTIVE_OTHER = 9, // invalid number
AR_SIT_UNKNOWN   = 10, // normal unspecified
AR_IN_QUEUE      = 11, // call still in queue - for
                    // Delivered Event only
AR_SERVICE_OBSERVER= 12 // service observer connected
} ATReasonCode_t
```

Charge Advice Event (Private)

Direction: Switch to Client

Event: *CSTAPrivateStatusEvent*

Private Data Event: *ATTChargeAdviceEvent*

Service Parameters: *monitorCrossRefID*

Private Parameters: *connection, calledDevice, chargingDevice, trunkGroup, trunkMember, chargeType, charge, error*

Functional Description:

This event reports the charging units for an outbound call to a trunk group (or all trunk groups) monitoring, a station monitoring, or a call monitoring session. This event is available only if trunk group (or all trunk groups) monitoring is requested to the switch for turning the Charge Advice feature on.

Service Parameters:

monitorCrossRefID [mandatory] Contains the handle to the monitor request for which this event is reported.

Private Parameters:

connection [mandatory] Specifies the connectionID of the trunk party that generates the charge event. The deviceID is null if split charge is reported due to a conference or transfer.

calledDevice [mandatory] Specifies the external device that was dialed or requested. This number does not include ARS, FAC, or TAC.

chargingDevice [mandatory] Specifies the local device that added the trunk group member to the call or an external party if the ISDN-PRI (or R2MFC) calling party number of the caller is available. If no local party is involved, and no calling party is available for an external call, then the TAC of the trunk used on the incoming call will be present. This number indicates to the application the number that may be used at the device that is being charged. Note that this number is not always identical to the CPN or SID that is provided in other event reports reporting on the same call.

trunkGroup [mandatory] Specifies the trunk group receiving the charge. The number provided correspond to the number used in switch administration, and is not the Trunk Access Code.

trunkMember [mandatory] Specifies the member of the trunk group receiving the charge.

chargeType [mandatory] Indicates the charge type provide by the network. Valid types are:

- CT_INTERMEDIATE_CHARGE — This is a charge sent by the trunk while the call is active. The charge amounts reported are cumulative. If a call receives two or more consecutive intermediate charges, then the amount from the last intermediate charge replaces the amount(s) of the previous intermediate charges. The amounts are not added to produce a total charge.
- CT_FINAL_CHARGE — This charge is sent by the trunk when a call is dropped. If call CDR outgoing call splitting is not enabled, then the final charge reflects the charge for the entire call.

- CT_SPLIT_CHARGE — CDR outgoing call splitting is used to divide the charge for a call among different users. For example, if an outgoing call is placed by one station and transferred to a second station, and if CDR call splitting is enabled, then CDR and the Charge Advice Events would charge the first station up to the time of the transfer, and the second station after that. A split charge reflects the charge for the call up to the time the split charge is sent (starting at the beginning of the call, or at the previous split charge). Any Charge Advice Event received after a split charge will reflect only that portion of the charge that took place after the split charge. If split charges are received for a call, then the total charge for the call can be computed by adding the split charges and the final charge.

charge

[mandatory] Specifies the amount of charging units.

error

[optional — supported] Indicates a possible error in the charge amount and the reason for the error. It will appear only if there is an error.

- CE_NONE — no error
- CE_NO_FINAL_CHARGE — network failed to provide a final charge for the call (CS3/38)
- CE_LESS_FINAL_CHARGE — final charge provided by the network is less than a previous charge (CS3/38)
- CE_CHARGE_TOO_LARGE — charge provided by the network is too large (CS3/38)
- CE_NETWORK_BUSY — too many calls are waiting for their final charge from the network (CS3/22)

Detailed Information:

- Charge Advice Event Feature — This feature must be turned on via `cstaMonitorDevice()` with `attMonitorDevice()`.
- Feature Availability — This feature is available starting with DEFINITY G3 Release 5.
- Trunk Group Administration — Only ISDN-PRI trunk groups that have Charge Advice set to “during-on-request” or “automatic” on the switch will receive Charge Advice Events.
- More Than 100 Calls in Call Clearing State — If more than 100 calls are in a call clearing state waiting for charging information, the oldest record will not receive final charge information. In this case a value of 0 and a cause value of CE_NETWORK_BUSY will be reported.

Syntax

```
#include <acs.h>
#include <csta.h>

// CSTAPrivateStatusEvent

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTAUNSOLICITED
    EventType_t eventType; // CSTA_PRIVATE_STATUS
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    struct
    {
        CSTAMonitorCrossRefID_t monitorCrossRefId;
        union
        {
            CSTAPrivateStatusEvent_t privateStatus;
        } cstaUnsolicited;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAPrivateStatusEvent_t
{
    Nulltype null;
} CSTAPrivateStatusEvent_t;
```

Private Parameter Syntax

If private data accompanies a CSTAPrivateStatusEvent, then the private data would be stored in the location that the application specified as the privateData parameter in the acsGetEventBlock() or acsGetEventPoll() request. If the privateData pointer is set to NULL in these requests, then the CSTAPrivateStatusEvent does not deliver private data to the application. If the acsGetEventBlock() or acsGetEventPoll() returns a Private Data length of 0, then no private data is provided with this event.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTChargeAdviceEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_t    eventType;    // ATT_CHARGE_ADVICE
    union
    {
        ATTChargeAdviceEvent_t    chargeAdviceEvent;
    } u;
} ATTEvent_t;

typedef struct ATTChargeAdviceEvent_t
{
    ConnectionID_t    connection;
    DeviceID_t        calledDevice;
    DeviceID_t        chargingDevice;
    DeviceID_t        trunkGroup;
    DeviceID_t        trunkMember;
    ATTChargeType_t   chargeType;
    long              charge;
    ATTChargeError_t   error;
} ATTChargeAdviceEvent_t;

typedef enum ATTChargeType_t
{
    CT_INTERMEDIATE_CHARGE    = 1,
    CT_FINAL_CHARGE           = 2,
    CT_SPLIT_CHARGE           = 3
} ATTChargeType_t;

typedef enum ATTChargeError_t
{
    CE_NONE                    = 0,
    CE_NO_FINAL_CHARGE         = 1,
    CE_LESS_FINAL_CHARGE      = 2,
    CE_CHARGE_TOO_LARGE       = 3,
    CE_NETWORK_BUSY           = 4
} ATTChargeError_t;
```

Conferenced Event

Direction: Switch to Client

Event: *CSTAConferencedEvent*

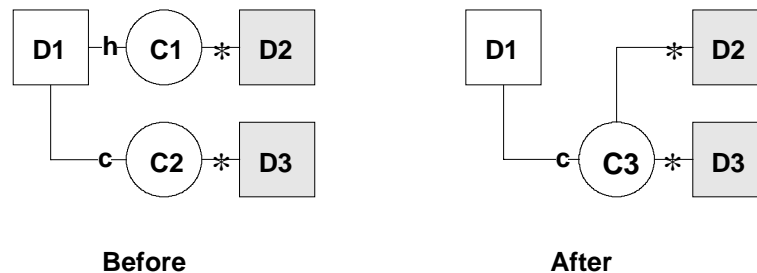
Private Data Event: *ATTConferencedEvent* (private data version 6), *ATTV5ConferencedEvent* (private data version 5), *ATTV4ConferencedEvent* (private data version 4), *ATTV3ConferencedEvent* (private data versions 2 and 3)

Service Parameters: *monitorCrossRefID*, *primaryOldCall*, *secondaryOldCall*, *confController*, *addedParty*, *conferenceConnections*, *localConnectionInfo*, *cause*

Private Parameters: *originalCallInfo*, *distributingDevice*, *ucid*

Functional Description:

The Conference Event Report indicates that two calls are conferenced (merged) into one, and no parties are removed from the resulting call in the process. The event may include up to six parties on the resulting call.



The Conferenced Event Report is generated for the following circumstances:

- When an on-PBX station completes a conference by pressing the “conference” button on the voice terminal.
- When an on-PBX station completes a conference after having activated the “supervisor assist” button on the voice set.
- When the on-PBX analog set user flashes the switch hook with one active call and one call on conference and/or transfer hold.
- When an application processor successfully completes a *cstaConferenceCall* request.
- When the “call park” feature is used in conjunction with the “conference” button on the voice set.

Service Parameters:

<i>monitorCrossRefID</i>	[[mandatory] Contains the handle to the monitor request for which this event is reported.
<i>primaryOldCall</i>	[mandatory] Specifies the callID of the call that was conferenced. This is usually the held call before the conference. This call is ended as a result of the conference.
<i>secondaryOldCall</i>	[mandatory] Specifies the callID of the call that was conferenced. This is usually the active call before the conference. This call was retained by the switch after the conference.
<i>contController</i>	[mandatory] Specifies the device that is controlling the conference. This is the device that set up the conference.
<i>addedParty</i>	[mandatory] Specifies the new conferenced-in device. <ul style="list-style-type: none"> ■ If the device is an on-PBX station, the extension is specified. ■ If the party is an off-PBX endpoint, then the deviceID is ID_NOT_KNOWN.¹ <p>There are call scenarios in which the conference operation joins multiple parties to a call. In such situations, the addedParty will be the extension for the last party to join the call.</p>
<i>conferenceConnections</i>	[optional — supported] Specifies a count of the number of devices and a list of connectionIDs and deviceIDs which resulted from the conference. <ul style="list-style-type: none"> ■ If a device is on-PBX, the extension is specified. The extension consists of station or group extensions. Group extensions are provided when the conference is to a group and the conference completes before the call is answered by one of the group members (TEG, PCOL, hunt group, or VDN extension). It may contain alerting extensions. ■ The static deviceID of a queued endpoint is set to the split extension of the queue. ■ If a party is off-PBX, then its static device identifier or its previously assigned trunk identifier is specified.
<i>localConnectionInfo</i>	[optional — supported] Specifies the local connection state as perceived by the monitored device on this call. This is provided for the cstaMonitorDevice requests only. A value of CS_NONE indicates that the local connection state is unknown.
<i>cause</i>	[optional — limited supported] Specifies the reason for this event.

- EC_PARK — A call conference was performed for parking a call rather than a true call conference operation.
- EC_ACTIVE_MONITOR — This is the cause value if the Single Step Conference request is for PT_ACTIVE. For details, see “Single Step Conference Call Service (Private Data Version 5 and Later)” in Chapter 4.
- EC_SILENT_MONITOR — This is the cause value if the Single Step Conference request is for PT_SILENT. For details, see “Single Step Conference Call Service (Private Data Version 5 and Later)” in Chapter 4.

-
1. This endpoint's trunk identifier is included in the conferenceConnections list, but not in this parameter.

Private Parameters:

originalCallInfo [optional] specifies the original call information. This parameter is sent with this event for the resulting newCall of a cstaConferenceCall request or the retained call of a (manual) conference call operation. The calls being conferenced must be known to the G3PD via the Call Control Services or Monitor Services.

⇒ NOTE:

For a cstaConferenceCall, the originalCallInfo includes the original call information originally received by the heldCall specified in the cstaConferenceCall request. For a manual call conference, the originalCallInfo includes the original call information originally received by the primaryOldCall specified in the event report.

The original call information includes:

- **reason** — the reason for the originalCallInfo. The following reasons are supported:
 - OR_NONE — no originalCallInfo provided
 - OR_CONFERENCED — call conferenced
- **callingDevice** — the original callingDevice received by the heldCall or the primaryOldCall. This parameter is always provided.
- **calledDevice** — the original calledDevice received by the heldCall or the primaryOldCall. This parameter is always provided.
- **trunk** — the original trunk group received by the heldCall or the primaryOldCall. This parameter is supported by private data versions 2, 3, and 4.
- **trunkGroup** — the original trunkGroup received by the heldCall or the primaryOldCall. This parameter is supported by private data version 5 and later only.
- **trunkMember** (G3V4 switches and later) — the original trunkMember received by the heldCall or the primaryOldCall.
- **lookaheadInfo** — the original lookaheadInfo received by the heldCall or the primaryOldCall.
- **userEnteredCode** — the original userEnteredCode received by the heldCall or the primaryOldCall call.

- **userInfo** — the original userInfo received by the heldCall or the primaryOldCall call.

Prior to G3V8, the maximum length of userInfo was 32 bytes. Beginning with G3V8, the maximum length of userInfo is increased to 96 bytes.

⇒ NOTE:

An application using private data version 5 and earlier can only receive a maximum of 32-byte data for userInfo, regardless of the size data that is sent by the switch.

- **ucid** — the original ucid of the call. This parameter is supported by private data version 5 and later only.
- **callOriginatorInfo** — the original callOriginatorInfo received by the activeCall. This parameter is supported by private data version 5 and later only.
- **flexibleBilling** — the original flexibleBilling information of the call. This parameter is supported by private data version 5 and later only.

See the “Delivered Event” section in this chapter for the details of these parameters.

distributingDevice	[optional] Specifies the original distributing device of the call before the call is conferenced. See the “Delivered Event” section in this chapter for details on the distributingDevice parameter. This parameter is supported by private data version 4 and later
ucid	[optional] Specifies the Universal Call ID (UCID) of the resulting newCall. The UCID is a unique call identifier across switches and the network. A valid UCID is a null-terminated ASCII character string. If there is no UCID associated with this call, the ucid contains the ATT_NULL_UCID (a 20-character string of all zeros). This parameter is supported by private data version 5 and later only.
trunkList	[optional] Specifies a list of up to 5 trunk groups and trunk members. This parameter is supported by private data version 6 and later only. The following options are supported: <ul style="list-style-type: none"> ■ count — The count of the connected parties on the call. ■ trunks — An array of 5 trunk group and trunk member IDs, one for each connected party. The following options are supported:

- connection — The connection ID of one of the parties on the call.
- trunkGroup — The trunk group of the party referenced by connection.
- trunkMember — The trunk member of the party referenced by connection.

Detailed Information:

See the “Event Report Detailed Information” section in this chapter.

- The originalCallInfo includes the original call information originally received by the call that is ended (this is usually, but not always, the held call) as the result of the conference.

The following special rules apply:

- If the Conferenced Event was a result of a cstaConferenceCall request, the originalCallInfo and the distributingDevice sent with this Conferenced Event is from the heldCall in the cstaConferenceCall request. Thus the application can control what originalCallInfo and distributingDevice to be sent in a Conferenced Event by putting the original call on hold and specifying it as the heldCall in the cstaConferenceCall request. The primaryOldCall (the call ended as the result of the cstaConferenceCall) is usually the heldCall, but it can be the activeCall.
- If the Conferenced Event was a result of a manual conference, the originalCallInfo and the distributingDevice sent with this Conferenced Event is from the primaryOldCall of the event. Thus the application does not have control of what originalCallInfo and the distributingDevice to be sent in the Conferenced Event. The primaryOldCall (the call ended as the result of the manual conference operation) is usually the held call, but it can be the active call.

Syntax

```

#include <acs.h>
#include <csta.h>

// CSTAConferencedEvent

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTAUNSOLICITED
    EventType_t eventType; // CSTA_CONFERENCED
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            CSTAMonitorCrossRefID_t monitorCrossRefId;
            union
            {
                CSTAConferencedEvent_t conferenced;
            } u;
        } cstaUnsolicited;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAConferencedEvent_t {
    ConnectionID_t primaryOldCall;
    ConnectionID_t secondaryOldCall;
    SubjectDeviceID_t confController;
    SubjectDeviceID_t addedParty;
    ConnectionList_t conferenceConnections;
    LocalConnectionState_t localConnectionInfo;
    CSTAEventCause_t cause;
} CSTAConferencedEvent_t;

typedef struct Connection_t {
    ConnectionID_t party;
    SubjectDeviceID_t staticDevice;
} Connection_t;

typedef struct ConnectionList_t {
    int count;
    Connection_t *connection;
} ConnectionList_t;

```

Private Data Version 6 Syntax

If private data accompanies a CSTAConferencedEvent, then the private data would be stored in the location that the application specified as the privateData parameter in the acsGetEventBlock() or acsGetEventPoll() request. If the privateData pointer is set to NULL in these requests, then the CSTAConferencedEvent does not deliver private data to the application. If acsGetEventBlock() or acsGetEventPoll() returns a Private Data length of 0, then no private data is provided with this event.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTConferencedEvent - CSTA Unsolicited Event Private Data

typedef struct ATTEvent_t
{
    ATTEventType_teventType;// ATT_CONFERENCED
    union
    {
        ATTConferencedEvent_tconferencedEvent;
    } u;
} ATTEvent_t;

typedef struct ATTConferencedEvent_t
{
    ATTOriginalCallInfo_t    originalCallInfo;
    CalledDeviceID_t        distributingDevice;
    ATTUCID_t                ucid;
} ATTConferencedEvent_t;

typedef struct ATTOriginalCallInfo_t
{
    ATTReasonForCallInfo_t    reason;
    CallingDeviceID_t        callingDevice;
    CalledDeviceID_t        calledDevice;
    DeviceID_t                trunkGroup;
    DeviceID_t                trunkMember;
    ATTLookaheadInfo_t        lookaheadInfo;
    ATTUserEnteredCode_t        userEnteredCode;
    ATTUserToUserInfo_t        userInfo;
    ATTUCID_t                ucid;
    ATTCallOriginatorInfo_t    callOriginatorInfo;
    Boolean                    flexibleBilling;
} ATTOriginalCallInfo_t;
```

Private Data Version 6 Syntax (Continued)

```

typedef enum ATTReasonForCallInfo_t
{
    OR_NONE                = 0, // indicates not present
    OR_CONSULTATION        = 1,
    OR_CONFERENCED         = 2,
    OR_TRANSFERRED         = 3,
    OR_NEW_CALL             = 4
} ATTReasonForCallInfo_t;

typedef ExtendedDeviceID_t  CallingDeviceID_t;

typedef ExtendedDeviceID_t  CalledDeviceID_t;

typedef struct ATTLookaheadInfo_t
{
    ATTInterflow_t          type;
    ATTPriority_t           priority;
    short                   hours;
    short                   minutes;
    short                   seconds;
    DeviceID_t              sourceVDN;
    ATTUnicodeDeviceID_t    uSourceVDN; // sourceVDN in Unicode
} ATTLookaheadInfo_t;

typedef enum ATTInterflow_t
{
    LAI_NO_INTERFLOW       = -1, // indicates info not present
    LAI_ALL_INTERFLOW      = 0,
    LAI_THRESHOLD_INTERFLOW= 1,
    LAI_VECTORIZING_INTERFLOW= 2
} ATTInterflow_t;

typedef enum ATTPriority_t
{
    LAI_NOT_IN_QUEUE       = 0,
    LAI_LOW                 = 1,
    LAI_MEDIUM              = 2,
    LAI_HIGH                = 3,
    LAI_TOP                 = 4
} ATTPriority_t;

typedef unsigned short ATTUnicodeDeviceID_t[64];

typedef struct ATTUserEnteredCode_t
{
    ATTUserEnteredCodeType_t    type;
    ATTUserEnteredCodeIndicator_t indicator;
    char                        data[ATT_MAX_USER_CODE];
    DeviceID_t                  collectVDN;
} ATTUserEnteredCode_t;

```

Private Data Version 6 Syntax (Continued)

```
typedef enum ATTUserEnteredCodeType_t
{
    UE_NONE          = -1, // indicates not specified
    UE_ANY           = 0,
    UE_LOGIN_DIGITS  = 2,
    UE_CALL_PROMPTER = 5,
    UE_DATA_BASE_PROVIDED = 17,
    UE_TONE_DETECTOR = 32
} ATTUserEnteredCodeType_t;

typedef enum ATTUserEnteredCodeIndicator_t
{
    UE_COLLECT = 0,
    UE_ENTERED = 1
} ATTUserEnteredCodeIndicator_t;

typedef struct ATTUserToUserInfo_t
{
    ATTUUIProtocolType_t type;
    struct {
        short          length; // 0 indicates UUI not
                               // present
        unsigned char  value[ATT_MAX_USER_INFO];
    } data;
} ATTUserToUserInfo_t;

typedef enum ATTUUIProtocolType_t
{
    UUI_NONE          = -1, // indicates not specified
    UUI_USER_SPECIFIC = 0, // user-specific
    UUI_IA5_ASCII     = 4 // null terminated ascii
                          // character string
} ATTUUIProtocolType_t;

typedef char ATTUCID_t[64];

typedef struct ATTCallOriginatorInfo_t
{
    Boolean hasInfo; // if FALSE, no callOriginatorType
    short    callOriginatorType;
} ATTCallOriginatorInfo_t;
```

Private Data Version 5 Syntax

If private data accompanies a CSTAConferencedEvent, then the private data would be stored in the location that the application specified as the privateData parameter in the acsGetEventBlock() or acsGetEventPoll() request. If the privateData pointer is set to NULL in these requests, then the CSTAConferencedEvent does not deliver private data to the application. If acsGetEventBlock() or acsGetEventPoll() returns a Private Data length of 0, then no private data is provided with this event.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV5ConferencedEvent - CSTA Unsolicited Event Private Data

typedef struct ATTEvent_t
{
    ATTEventType_teventType;// ATT_CONFERENCED
    union
    {
        ATTV5ConferencedEvent_tconferencedEvent;
    } u;
} ATTEvent_t;

typedef struct ATTV5ConferencedEvent_t
{
    ATTV5OriginalCallInfo_t originalCallInfo;
    CalledDeviceID_t        distributingDevice;
    ATTUCID_t               ucid;
} ATTV5ConferencedEvent_t;

typedef struct ATTV5OriginalCallInfo_t
{
    ATTReasonForCallInfo_t reason;
    CallingDeviceID_t      callingDevice;
    CalledDeviceID_t       calledDevice;
    DeviceID_t             trunkGroup;
    DeviceID_t             trunkMember;
    ATTLookaheadInfo_t     lookaheadInfo;
    ATTUserEnteredCode_t   userEnteredCode;
    ATTV5UserToUserInfo_t  userInfo;
    ATTUCID_t              ucid;
    ATTV5CallOriginatorInfo_tcallOriginatorInfo;
    Boolean                 flexibleBilling;
} ATTV5OriginalCallInfo_t;
```

Private Data Version 5 Syntax (Continued)

```
typedef enum ATTReasonForCallInfo_t
{
    OR_NONE                = 0, // indicates not present
    OR_CONSULTATION        = 1,
    OR_CONFERENCED         = 2,
    OR_TRANSFERRED         = 3,
    OR_NEW_CALL            = 4
} ATTReasonForCallInfo_t;

typedef ExtendedDeviceID_t CallingDeviceID_t;

typedef ExtendedDeviceID_t CalledDeviceID_t;

typedef struct ATTLookaheadInfo_t
{
    ATTInterflow_t        type;
    ATTPriority_t         priority;
    short                 hours;
    short                 minutes;
    short                 seconds;
    DeviceID_t            sourceVDN;
    ATTUnicodeDeviceID_t uSourceVDN; // sourceVDN in Unicode
} ATTLookaheadInfo_t;

typedef enum ATTInterflow_t
{
    LAI_NO_INTERFLOW      = -1, // indicates info not present
    LAI_ALL_INTERFLOW     = 0,
    LAI_THRESHOLD_INTERFLOW = 1,
    LAI_VECTORING_INTERFLOW = 2
} ATTInterflow_t;

typedef enum ATTPriority_t
{
    LAI_NOT_IN_QUEUE      = 0,
    LAI_LOW                = 1,
    LAI_MEDIUM            = 2,
    LAI_HIGH               = 3,
    LAI_TOP                = 4
} ATTPriority_t;

typedef unsigned short ATTUnicodeDeviceID_t[64];

typedef struct ATTUserEnteredCode_t
{
    ATTUserEnteredCodeType_t    type;
    ATTUserEnteredCodeIndicator_t indicator;
    char                        data[33];
    DeviceID_t                  collectVDN;
} ATTUserEnteredCode_t;
```


Private Data Version 5 Syntax (Continued)

```

typedef enum ATTUserEnteredCodeType_t
{
    UE_NONE           = -1, // indicates not specified
    UE_ANY            = 0,
    UE_LOGIN_DIGITS   = 2,
    UE_CALL_PROMPTER = 5,
    UE_DATA_BASE_PROVIDED= 17,
    UE_TONE_DETECTOR  = 32
} ATTUserEnteredCodeType_t;

typedef enum ATTUserEnteredCodeIndicator_t
{
    UE_COLLECT = 0,
    UE_ENTERED = 1
} ATTUserEnteredCodeIndicator_t;

typedef struct ATTV5UserToUserInfo_t
{
    ATTUIProtocolType_ttype;
    struct {
        short          length; // 0 indicates UII not
                               // present
        unsigned char  value[33];
    } data;
} ATTV5UserToUserInfo_t;

typedef enum ATTUIProtocolType_t
{
    UII_NONE           = -1, // indicates not specified
    UII_USER_SPECIFIC = 0, // user-specific
    UII_IA5_ASCII      = 4 // null terminated ascii
                          // character string
} ATTUIProtocolType_t;

typedef char ATTUCID_t[64];

typedef struct ATTCallOriginatorInfo_t
{
    Boolean hasInfo; // if FALSE, no callOriginatorType
    short    callOriginatorType;
} ATTCallOriginatorInfo_t;

```

Private Data Version 4 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV4ConferencedEvent - CSTA Unsolicited Event Private
Data

typedef struct ATTEvent_t
{
    ATTEventType_t eventType; // ATTV4_CONFERENCED
    union
    {
        ATTV4ConferencedEvent_t tv4conferencedEvent;
    } u;
} ATTEvent_t;

typedef struct ATTV4ConferencedEvent_t
{
    ATTV4OriginalCallInfo_t originalCallInfo;
    CalledDeviceID_t         distributingDevice;
} ATTV4ConferencedEvent_t;

typedef struct ATTV4OriginalCallInfo_t
{
    ATTReasonForCallInfo_t reason;
    CallingDeviceID_t      callingDevice;
    CalledDeviceID_t       calledDevice;
    DeviceID_t             trunk;
    DeviceID_t             trunkMember;
    ATTV4LookaheadInfo_t  lookaheadInfo;
    ATTUserEnteredCode_t  userEnteredCode;
    ATTV5UserToUserInfo_t userInfo;
} ATTV4OriginalCallInfo_t;

typedef enum ATTReasonForCallInfo_t
{
    OR_NONE = 0, // indicates not present
    OR_CONSULTATION = 1,
    OR_CONFERENCED = 2,
    OR_TRANSFERRED = 3,
    OR_NEW_CALL = 4
} ATTReasonForCallInfo_t;

typedef ExtendedDeviceID_t CallingDeviceID_t;

typedef ExtendedDeviceID_t CalledDeviceID_t;
```

Private Data Version 4 Syntax (Continued)

```

typedef struct ATTV4LookaheadInfo_t
{
    ATTInterflow_t    type;
    ATTPriority_t     priority;
    short              hours;
    short              minutes;
    short              seconds;
    DeviceID_t        sourceVDN;
} ATTV4LookaheadInfo_t;

typedef enum ATTInterflow_t
{
    LAI_NO_INTERFLOW = -1,    // indicates info not present
    LAI_ALL_INTERFLOW= 0,
    LAI_THRESHOLD_INTERFLOW= 1,
    LAI_VECTORING_INTERFLOW= 2
} ATTInterflow_t;

typedef enum ATTPriority_t
{
    LAI_NOT_IN_QUEUE      = 0,
    LAI_LOW                = 1,
    LAI_MEDIUM            = 2,
    LAI_HIGH               = 3,
    LAI_TOP                = 4
} ATTPriority_t;

typedef struct ATTUserEnteredCode_t
{
    ATTUserEnteredCodeType_t    type;
    ATTUserEnteredCodeIndicator_t indicator;
    char                          data[33];
    DeviceID_t                    collectVDN;
} ATTUserEnteredCode_t;

typedef enum ATTUserEnteredCodeType_t
{
    UE_NONE                = -1,    // indicates not specified
    UE_ANY                  = 0,
    UE_LOGIN_DIGITS       = 2,
    UE_CALL_PROMPTER     = 5,
    UE_DATA_BASE_PROVIDED= 17,
    UE_TONE_DETECTOR     = 32
} ATTUserEnteredCodeType_t;

typedef enum ATTUserEnteredCodeIndicator_t
{
    UE_COLLECT            = 0,
    UE_ENTERED           = 1
} ATTUserEnteredCodeIndicator_t;

```

Private Data Version 4 Syntax (Continued)

```
typedef struct ATTV5UserToUserInfo_t
{
    ATTUUIProtocolType_t type;
    struct {
        short          length; // 0 indicates UUI
                                // not present
        unsigned charvalue[33];
    } data;
} ATTV5UserToUserInfo_t;

typedef enum ATTUUIProtocolType_t
{
    UUI_NONE           = -1, // indicates not specified
    UUI_USER_SPECIFIC = 0, // user-specific
    UUI_IA5_ASCII      = 4 // null terminated ascii
                        // character string
} ATTUUIProtocolType_t;
```

Private Data Versions 2 and 3 Syntax

```

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV3ConferencedEvent - CSTA Unsolicited Event Private
Data

typedef struct ATTEvent_t
{
    ATTEventType_t eventType; // ATTV3_CONFERENCED
    union
    {
        ATTV3ConferencedEvent_t tv3conferencedEvent;
    } u;
} ATTEvent_t;

typedef struct ATTV3ConferencedEvent_t
{
    ATTV4OriginalCallInfo_t originalCallInfo;
} ATTV3ConferencedEvent_t;

typedef struct ATTV4OriginalCallInfo_t
{
    ATTReasonForCallInfo_t reason;
    CallingDeviceID_t callingDevice;
    CalledDeviceID_t calledDevice;
    DeviceID_t trunk;
    DeviceID_t trunkMember;
    ATTV4LookaheadInfo_t lookaheadInfo;
    ATTUserEnteredCode_t userEnteredCode;
    ATTV5UserToUserInfo_t userInfo;
} ATTV4OriginalCallInfo_t;

typedef enum ATTReasonForCallInfo_t
{
    OR_NONE = 0, // indicates not present
    OR_CONSULTATION = 1,
    OR_CONFERENCED = 2,
    OR_TRANSFERRED = 3,
    OR_NEW_CALL = 4
} ATTReasonForCallInfo_t;

typedef ExtendedDeviceID_t CallingDeviceID_t;

typedef ExtendedDeviceID_t CalledDeviceID_t;

```

Private Data Versions 2 and 3 Syntax (Continued)

```
typedef struct ATTV4LookaheadInfo_t
{
    ATTInterflow_t    type;
    ATTPriority_t     priority;
    short              hours;
    short              minutes;
    short              seconds;
    DeviceID_t        sourceVDN;
} ATTV4LookaheadInfo_t;

typedef enum ATTInterflow_t
{
    LAI_NO_INTERFLOW= -1, // indicates info not present
    LAI_ALL_INTERFLOW= 0,
    LAI_THRESHOLD_INTERFLOW= 1,
    LAI_VECTORING_INTERFLOW= 2
} ATTInterflow_t;

typedef enum ATTPriority_t
{
    LAI_NOT_IN_QUEUE = 0,
    LAI_LOW           = 1,
    LAI_MEDIUM        = 2,
    LAI_HIGH          = 3,
    LAI_TOP           = 4
} ATTPriority_t;

typedef struct ATTUserEnteredCode_t
{
    ATTUserEnteredCodeType_t    type;
    ATTUserEnteredCodeIndicator_t indicator;
    char                          data[33];
    DeviceID_t                    collectVDN;
} ATTUserEnteredCode_t;

typedef enum ATTUserEnteredCodeType_t
{
    UE_NONE           = -1, // indicates not specified
    UE_ANY            = 0,
    UE_LOGIN_DIGITS  = 2,
    UE_CALL_PROMPTER = 5,
    UE_DATA_BASE_PROVIDED= 17,
    UE_TONE_DETECTOR = 32
} ATTUserEnteredCodeType_t;

typedef enum ATTUserEnteredCodeIndicator_t
{
    UE_COLLECT    = 0,
    UE_ENTERED    = 1
} ATTUserEnteredCodeIndicator_t;
```

Private Data Versions 2 and 3 Syntax (Continued)

```

typedef struct ATTV5UserToUserInfo_t
{
    ATTUUIProtocolType_t type;
    struct {
        short          length; // 0 indicates UUI
                               // not present
        unsigned char  value[33];
    } data;
} ATTV5UserToUserInfo_t;

typedef enum ATTUUIProtocolType_t
{
    UUI_NONE           = -1, // indicates not specified
    UUI_USER_SPECIFIC = 0, // user-specific
    UUI_IA5_ASCII      = 4 // null terminated ascii
                        // character string
} ATTUUIProtocolType_t;

```

Connection Cleared Event

Direction: Switch to Client

Event: CSTAConnectionClearedEvent

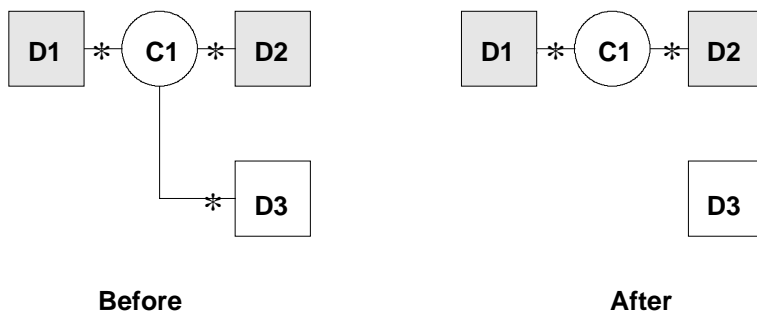
Private Data Event: ATTConnectionClearedEvent (private data version 6),
ATTV5ConnectionClearedEvent (private data version 2, 3, 4 and 5)

Service Parameters: monitorCrossRefID, droppedConnection,
releasingDevice, localConnectionInfo, cause

Private Parameters: userInfo

Functional Description:

The Connection Cleared Event Report indicates that a device in a call disconnects or is dropped. It does not indicate that a transferring device has left a call in the act of transferring that call.



A Connection Cleared Event Report is generated in the following cases:

- A simulated bridged appearance is dropped when one member drops.
- When an on-PBX party drops from a call.
- When an off-PBX party drops and the ISDN-PRI receives a disconnect message.
- When an off-PBX party drops and the non-ISDN-PRI trunk detects a drop.

A Connection Cleared Event Report is not generated in the following cases:

- A party drops as a result of a transfer operation.
- A split or vector announcement drops.
- Attendant drops a call, if the call was received through the attendant group (0).
- A cstaMakePredictiveCall call is dropped during the call classification stage. (A Call Cleared Event Report is generated instead.)
- A call is delivered to an agent and de-queued from multiple splits as part of vector processing.

This event report is not generated for the last disconnected party on a call for a `cstaMonitorCallsViaDevice` request. In this case, a Call Cleared Event Report is generated instead. This event is the last event of a call for a `cstaMonitorDevice` request.

Service Parameters:

- monitorCrossRefID*** [mandatory] Contains the handle to the monitor request for which this event is reported.
- droppedConnection*** [mandatory] Specifies the connection that has been dropped from the call.
- releasingDevice*** [mandatory] Specifies the dropped device.
- If the device is on-PBX, then the extension is specified (primary extension for TEGs, PCOLs, bridging).
 - If a party is off-PBX, then its static device identifier or its previously assigned trunk identifier is specified.
- localConnectionInfo*** [optional — supported] Specifies the local connection state as perceived by the monitored device on this call. This is provided for *cstaMonitorDevice* requests only. A value of *CS_NONE* indicates that the local connection state is unknown.
- cause*** [optional — supported] Specifies a cause when the call is not terminated normally. *EC_NONE* is specified for normal call termination.
- *EC_BUSY* — Device busy.
 - *EC_CALL_CANCELLED* — Call rejected or canceled.
 - *EC_DEST_NOT_OBTAINABLE* — Called device is not reachable or wrong number is called
 - *EC_CALL_NOT_ANSWERED* — Called device not responding or call not answered (*maxRings* has timed out) for a *MakePredictiveCall*.
 - *EC_NETWORK_CONGESTION* — Network congestion or channel is unacceptable.
 - *EC_RESOURCES_NOT_AVAILABLE* — No circuit or channel is available.
 - *EC_TRANSFER* — Call merged due to transfer or conference.
 - *EC_REORDER_TONE* — Intercept SIT treatment - Number changed.
 - *EC_VOICE_UNIT_INITIATOR* — Answer machine is detected for a *MakePredictiveCall*.

Private Parameters:

userInfo

[optional] Contains user-to-user information. This parameter allows an application to associate caller information, up to 32 or 96 bytes, with a call. This information may be a customer number, credit card number, alphanumeric digits, or a binary string. It is propagated with the call when the call is dropped by a `cstaClearConnection` with a `userInfo` and passed to an application in the Connection Cleared Event Report.

Prior to G3V8, the maximum length of `userInfo` was 32 bytes. Beginning with G3V8, the maximum length of `userInfo` is increased to 96 bytes.

NOTE:

An application using private data version 5 and earlier can only receive a maximum of 32-byte data for `userInfo`, regardless of the size data that is sent by the switch.

The following UUI protocol types are supported:

- `UUI_NONE` — There is no data provided in the data parameter.
- `UUI_USER_SPECIFIC` — The content of the data parameter is a binary string. The correct size (maximum of 32 or 96 bytes) of data must be specified in the size parameter.
- `UUI_IA5_ASCII` — The content of the data parameter must be a null-terminated IA5 (ASCII) character string. The correct size (maximum of 32 or 96 bytes excluding the null terminator) of data must be specified in the size parameter.

Detailed Information:

See the “Event Report Detailed Information” section in this chapter.

Syntax

```
#include <acs.h>
#include <csta.h>

// CSTAConnectionClearedEvent

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTAUNSOLICITED
    EventType_t eventType; // CSTA_CONNECTION_CLEARED
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            CSTAMonitorCrossRefID_t monitorCrossRefId;
            union
            {
                CSTAConnectionClearedEvent_t
connectionCleared;
            } u;
        } cstaUnsolicited;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAConnectionClearedEvent_t
{
    ConnectionID_t droppedConnection;
    SubjectDeviceID_t releasingDevice;
    SubjectDeviceID_t localConnectionInfo;
    CSTAEventCause_t cause;
} CSTAConnectionClearedEvent_t;
```

Private Parameter 6 Syntax

If private data accompanies a `CSTAConnectionClearedEvent`, then the private data would be stored in the location that the application specified as the `privateData` parameter in the `acsGetEventBlock()` or `acsGetEventPoll()` request. If the `privateData` pointer is set to `NULL` in these requests, then the `CSTAConnectionClearedEvent` does not deliver private data to the application. If the `acsGetEventBlock()` or `acsGetEventPoll()` returns a Private Data length of 0, then no private data is provided with this event.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTConnectionClearedEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_t eventType; // ATT_CONNECTION_CLEARED
    union
    {
        ATTConnectionClearedEvent_t connectionCleared;
    } u;
} ATTEvent_t;

typedef struct ATTConnectionClearedEvent_t
{
    ATTUserToUserInfo_t userInfo;
} ATTConnectionClearedEvent_t;

typedef struct ATTUserToUserInfo_t {
    ATTUIProtocolType_t type;
    struct {
        short          length; // 0 indicates UII not present
        unsigned char  value[ATT_MAX_USER_INFO];
    } data;
} ATTUserToUserInfo_t;

typedef enum ATTUIProtocolType_t
{
    UII_NONE           = -1, // indicates not specified
    UII_USER_SPECIFIC  = 0, // user-specific
    UII_IA5_ASCII      = 4 // null terminated ascii
// character string
} ATTUIProtocolType_t
```

Private Data Version 2-5 Syntax

If private data accompanies a CSTAConnectionClearedEvent, then the private data would be stored in the location that the application specified as the privateData parameter in the acsGetEventBlock() or acsGetEventPoll() request. If the privateData pointer is set to NULL in these requests, then the CSTAConnectionClearedEvent does not deliver private data to the application. If the acsGetEventBlock() or acsGetEventPoll() returns a Private Data length of 0, then no private data is provided with this event.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV5ConnectionClearedEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_teventType;// ATTV5_CONNECTION_CLEARED
    union
    {
        ATTV5ConnectionClearedEvent_tconnectionCleared;
    } u;
} ATTEvent_t;

typedef struct ATTV5ConnectionClearedEvent_t
{
    ATTV5UserToUserInfo_tuserInfo;
} ATTV5ConnectionClearedEvent_t;

typedef structATTV5UserToUserInfo_t {
    ATTUUIProtocolType_t type;
    struct {
        short          length;// 0 indicates UUI not present
        unsigned char  value[33];
    } data;
} ATTV5UserToUserInfo_t;

typedef enum ATTUUIProtocolType_t
{
    UUI_NONE           = -1,// indicates not specified
    UUI_USER_SPECIFIC = 0,// user-specific
    UUI_IA5_ASCII     = 4 // null terminated ascii
// character string
} ATTUUIProtocolType_t
```

Delivered Event

Direction: Switch to Client

Event: CSTADeliveredEvent

Private Data Event: ATTDeliveredEvent (private data version 6), ATTV5DeliveredEvent (private data version 5), ATTV4DeliveredEvent (private data version 4), ATTV3DeliveredEvent (private data versions 2 and 3)
Service Parameters: monitorCrossRefID, connection, alertingDevice, callingDevice, calledDevice, lastRedirectionDevice, localConnectionInfo, cause

Private Parameters: deliveredType, trunk, trunkGroup, trunkMember, split, lookaheadInfo, userEnteredCode, userInfo, reason, originalCallInfo, distributingDevice, ucid, callOriginatorInfo, flexibleBilling

Functional Description:

The G3 switch reports two types of Delivered Event Reports (i.e., call delivered to station and call delivered to ACD/VDN). The type of the Delivered Event is specified in the ATTDeliveredEvent.

Call Delivered to a Station Device

A Delivered Event Report of this type indicates that “alerting” (tone, ring, etc.) is applied to a device or when the switch detects that “alerting” has been applied to a device.



Consecutive Delivered Event Reports are possible. Multiple Delivered Event Reports for multiple devices are also possible (e.g., a principal and its bridging users). The Delivered Event Report is not guaranteed for each call. The Delivered Event Report is not sent for calls that connect to announcements as a result of ACD split forced announcement or announcement vector commands.

The switch generates the Delivered Event Report when the following events occur.

- “Alerting” (tone, ring, etc.) is applied to a device or when the switch detects that “alerting” has been applied to a device.
- The originator of a cstaMakePredictiveCall call is an on-PBX station and ringing or zip tone is started.
- When a call is redirected to an off-PBX station and the ISDN ALERTing message is received from an ISDN-PRI facility.

- When a cstaMakePredictiveCall call is trying to reach an off-PBX station and the call classifier detects precise, imprecise, or special ringing.
- When a cstaMakeCall (or a cstaMakePredictiveCall) call is placed to an off-PBX station, and the ALERTing message is received from the ISDN-PRI facility.

When both a classifier and an ISDN-PRI facility report alerting on a call made by a cstaMakePredictiveCall request, then the first occurrence generates a Delivered Event Report; succeeding reports are not reported by the switch.

Consecutive Delivered Event Reports are possible in the following cases:

- A station is alerted first and the call goes to coverage: a Delivered Event Report is generated each time a new station is alerted.
- A principal and its bridging users are alerted: a Delivered Event Report is generated for the principal and for each bridged station alerted.
- A call is alerting a Terminating Extension Group (TEG); one report is sent for each TEG member alerted.
- A call is alerting a Personal Central Office Line (PCOL); one report is sent for each PCOL member is alerted.
- A call is alerting a coverage/answer point; one report is sent for each alerting member of the coverage answer group.
- A call is alerting a principal with SAC active; one report is sent for the principal and one or more are sent for the coverage points.

Call Delivered to an ACD Device

An ACD device can distribute calls within a switch. If an ACD device is called, normally the call will pass through the device, as the ACD call processing progresses, and eventually be delivered to a station device. Therefore, a call delivered to an ACD device will have multiple Delivered Event Reports before it connects.



There are two types of G3 devices that distribute calls, VDN and ACD split.

A Delivered Event Report of this type is generated when a call is delivered to an ACD device.

- Call Delivered to a VDN — This event is generated when a call is delivered to a monitored VDN.

- Call Delivered to an ACD Split — This event is generated when a call is delivered to a monitored ACD split. The report will be sent even if the ACD split is in night service or has call forwarding active.

A report will be generated for each `cstaMonitorCallsViaDevice` request that monitors an ACD device through which the call passes.

The Delivered Event Report is not sent for calls that connected to announcements as a result of ACD split forced announcement or announcement vector commands.

Service Parameters:

<i>monitorCrossRefID</i>	[mandatory] Contains the handle to the monitor request for which this event is reported.
<i>connection</i>	[mandatory] Specifies the endpoint that is alerting.
<i>alertingDevice</i>	[mandatory] Specifies the device that is alerting. <ul style="list-style-type: none"> ■ If the device being alerted is on-PBX, then the extension of the device is specified (primary extension for TEGs, PCOLs, bridging). ■ If a party is off-PBX, then its static device identifier or its assigned trunk identifier is specified. ■ If the call was delivered to a VDN or ACD split, the monitored object is specified.
<i>callingDevice</i>	[mandatory] Specifies the calling device. The following rules apply: <ul style="list-style-type: none"> ■ For internal calls — the originator’s extension. ■ For outgoing calls over PRI facilities¹ — “calling number” from the ISDN SETUP message or its assigned trunk identifier is specified. If the “calling number” does not exist, it is NULL. ■ For incoming calls over PRI facilities — “calling number” from the ISDN SETUP message or its assigned trunk identifier is specified. If the “calling number” does not exist, it is NULL. ■ For incoming calls over non-PRI facilities — the calling party number is generally not available. The assigned trunk identifier² is provided instead. ■ The trunk identifier is specified only when the calling party number is not available. ■ For calls originated at a bridged call appearance — the principal’s extension is specified. ■ There is a special case of a <i>cstaMakePredictiveCall</i> call being delivered to a split: in this case, the <i>callingDevice</i> contains the original digits (from the <i>cstaMakePredictiveCall</i> request) provided in the destination field.
<i>calledDevice</i>	[mandatory] Specifies the originally called device. The following rules apply: <ul style="list-style-type: none"> ■ For outgoing calls over PRI facilities — “called number” from the ISDN SETUP message is specified. If the “called number” does not exist (it is NULL), the <i>deviceIDStatus</i> is ID_NOT_KNOWN.

- For outgoing calls over non-PRI facilities — the deviceIDStatus is ID_NOT_KNOWN.
- For incoming calls over PRI facilities — “called number” from the ISDN SETUP message is specified.
- For incoming calls over non-PRI facilities — the principal extension is specified. It may be a group extension for TEG, hunt group, VDN. If the switch is administered to modify the DNIS digits, then the modified DNIS string is specified.
- For incoming calls to PCOL, the deviceIDStatus is ID_NOT_KNOWN.
- For incoming calls to a TEG (principal) group, the TEG group extension is specified.
- For incoming calls to a principal with bridges, the principal’s extension is specified.
- If the called device is on-PBX and the call did not come over a PRI facility, the extension of the party dialed is specified.

<i>lastRedirectionDevice</i>	[optional — limited support] Specifies the previous redirection/alerted device in the case where the call was redirected/diverted to the alertingDevice.
<i>localConnectionInfo</i>	[optional — supported] Specifies the local connection state as perceived by the monitored device on this call. This is provided for cstaMonitorDevice requests only. A value of CS_NONE means the local connection state is unknown.
<i>cause</i>	<p>[optional — supported] Specifies the cause for this event. The following causes are supported:</p> <p>The following four causes (i.e., EC_CALL_FORWARD, EC_CALL_FORWARD_ALWAYS, EC_CALL_FORWARD_BUSY, and EC_CALL_FORWARD_NO_ANSWER) are only available on a G3 PBX with G3V4 or later software. They have higher precedence than the other three causes (i.e., EC_KEY_CONFERENCE, EC_NEW_CALL, and EC_REDIRECTED). For example, if two causes apply to an event; one from the group with higher precedence (e.g., EC_CALL_FORWARD_ALWAYS) and one from the group with a lower precedence (e.g., EC_NEW_CALL), only the cause from the group with the higher precedence will apply.</p> <ul style="list-style-type: none"> ■ EC_CALL_FORWARD (G3V4 or later) — The call has been redirected via one of the following features:

- Send All Calls
- Cover All Calls
- Go to Cover active
- cstaDeflectCall
- EC_CALL_FORWARD_ALWAYS (G3V4 or later) — The call has been redirected via the Call Forwarding feature.
- EC_CALL_FORWARD_BUSY (G3V4 or later) — The call has been redirected for one of the following reasons:
 - Cover — principal busy
 - Cover — all call appearance busy
- EC_CALL_FORWARD_NO_ANSWER (G3V4 or later) — The call has been redirected because no answer from cover
- EC_KEY_CONFERENCE — Indicates that the event report occurred at a bridged device. This cause has higher precedence than the following two causes.
- EC_NEW_CALL — The call has not yet been redirected.
- EC_REDIRECTED — The call has been redirected.

-
1. For outgoing calls over non-PRI facilities, there is no Delivered Event Report. A Network Reached Event Report is generated instead.
 2. The trunk identifier is a dynamic device identifier and it cannot be used to access a trunk in the G3 switch.

Private Parameters:

<i>deliveredType</i>	[optional] Specifies the type of the Delivered Event: <ul style="list-style-type: none">■ DELIVERED_TO_ACD — This type indicates that the call is delivered to an ACD split or a VDN device and subsequent Delivered or other events (e.g., QUEUED) may be expected.■ DELIVERED_TO_STATION — This type indicates that the call is delivered to a station.■ DELIVERED_OTHER — This type is not in use.
<i>trunkGroup</i>	[optional] Specifies the trunk group number from which the call originated. Beginning with G3V8, trunk group number is provided regardless of whether the callingDevice is available. Prior to G3V8, trunk group number is provided only if the callingDevice is unavailable. This parameter is supported by private data version 5 and later only.
<i>trunk</i>	[optional] Specifies the trunk group number from which the call originated. Trunk group number is provided only if the callingDevice is unavailable. This parameter is supported by private data versions 2, 3, and 4 only.
<i>trunkMember</i>	[optional — limited supported] This parameter is supported beginning with G3V4. It specifies the trunk member number from which the call originated. Beginning with G3V8, trunk member number is provided regardless of whether the callingDevice is available. Prior to G3V8, trunk member number is provided only if the callingDevice is unavailable.
<i>split</i>	[optional] Specifies the ACD split extension to which the call is delivered. This parameter applies to DELIVERED_TO_STATION only.
<i>lookaheadinfo</i>	[optional] Specifies the lookahead interflow information received from the delivered call. Lookahead interflow is a G3 switch feature that routes some of the incoming calls from one switch to another so that they can be handled more efficiently and will not be lost. The switch that overflows the call provides the lookahead interflow information. A routing application may use the lookahead interflow information to determine the destination of the call. See the G3 Feature Description for more information about lookahead interflow. If the lookahead interflow type is set to "LAI_NO_INTERFLOW", no lookahead interflow private data is provided with this event.

userEnteredCode [optional] Specifies the code/digits that may have been entered by the caller through the G3 call prompting feature or the collected digits feature. If the userEnteredCode code is set to "UE_NONE", no userEnteredCode private data is provided with this event. See the "Detailed Information:" section for how to setup the switch and application for collecting userEnteredCode.

userInfo [optional] Contains user-to-user information. This parameter allows an application to associate caller information, up to 32 or 96 bytes, with a call. This information may be a customer number, credit card number, alphanumeric digits, or a binary string.

Prior to G3V8, the maximum length of userInfo was 32 bytes. Beginning with G3V8, the maximum length of userInfo is increased to 96 bytes.

⇒ NOTE:

An application using private data version 5 and earlier can only receive a maximum of 32-byte data for userInfo, regardless of the size data that is sent by the switch.

The following UUI protocol types are supported:

- UUI_NONE — There is no data provided in the data parameter.
- UUI_USER_SPECIFIC — The content of the data parameter is a binary string. The correct size (maximum of 32 or 96 bytes) of data must be specified in the size parameter.
- UUI_IA5_ASCII — The content of the data parameter must be a null-terminated IA5 (ASCII) character string. The correct size (maximum of 32 or 96 bytes excluding the null terminator) of data must be specified in the size parameter.

reason [optional] Specifies the reason of this event. The following reason is supported:

- AR_NONE— indicate no value specified for reason.
- AR_IN_QUEUE — When an already queued call reaches a converse vector step, the Delivered Event will include this reason code to inform the application that the call is still in queue. This reason applies to DELIVERED_TO_ACD only. Otherwise, this parameter will be set to AR_NONE.

originalCallInfo

[optional] Specifies the original call information. Note that information is not repeated in the originalCallInfo, if it is already reported in the CSTA service parameters or in the private data. For example, the callingDevice and calledDevice in the originalCallInfo will be NULL, if the callingDevice and the calledDevice in the CSTA service parameters are the original calling and called devices. Only when the original devices are different from the most recent callingDevice and calledDevice, the callingDevice and calledDevice in the originalCallInfo will be set. If the userEnteredCode in the private data is the original userEnteredCode, the userEnteredCode in the originalCallInfo will be UE_NONE. Only when new userEnteredCode is received and reported in the userEnteredCode, the originalCallInfo will have the original userEnteredCode

⇒ NOTE:

For the Delivered Event sent to the newCall of a Consultation Call, the originalCallInfo is taken from the activeCall specified in the Consultation Call request. Thus the application can pass the original call information between two calls. The calledDevice of the Consultation Call must reside on the same switch and must be monitored via the same Tserver.

The original call information includes:

- **reason** — the reason for the originalCallInfo. The following reasons are supported.
 - OR_NONE — no originalCallInfo provided
 - OR_CONSULTATION — consultation call
 - OR_CONFERENCED — call conferenced
 - OR_TRANSFERRED — call transferred
 - OR_NEW_CALL — new call
- **callingDevice** — the original callingDevice received by the activeCall.
- **calledDevice** — the original calledDevice received by the activeCall.
- **trunk** — the original trunk group received by the activeCall. This parameter is supported by private data version 2, 3, and 4.

- **trunkGroup** — the original trunkGroup received by the activeCall. This parameter is supported by private data version 5 and later only.
- **trunkMember** (G3V4 switches and later) — the original trunkMember received by the activeCall.
- **lookaheadInfo** — the original lookaheadInfo received by the activeCall.
- **userEnteredCode** — the original userEnteredCode received by the activeCall.
- **userInfo** — the original userInfo received by the activeCall.

Prior to G3V8, the maximum length of userInfo was 32 bytes. Beginning with G3V8, the maximum length of userInfo is increased to 96 bytes.

⇒ NOTE:

An application using private data version 5 and earlier can only receive a maximum of 32-byte data for userInfo, regardless of the size data that is sent by the switch.

- **ucid** — the original ucid of the call. This parameter is supported by private data version 5 and later only.
- **callOriginatorInfo** — the original callOriginatorInfo received by the activeCall. This parameter is supported by private data version 5 and later only.
- **flexibleBilling** — the original flexibleBilling information of the call. This parameter is supported by private data version 5 and later only.

distributingDevice

[optional] Specifies the ACD or VDN device that distributed the call to the agent station. This information is provided only when the call was processed by the switch ACD or Call Vectoring processing and is only sent for a station monitor (i.e., the delivery type is DELIVERED_TO_STATION). This parameter is supported by private data version 4 and later

⇒ NOTE:

The calledDevice specifies the originally called device. In most ACD call scenarios, calledDevice and distributingDevice have the same device ID. However, in call scenarios that involve call vectoring with the VDN Override feature turned on, calledDevice and distributingDevice may have different device IDs. Incoming calls arriving at the same calledDevice may be distributed to an agent via different call paths that have more than one VDN involved. If the VDN Override feature is used on the calledDevice, the distributingDevice specifies the VDN that distributes the call to the agent. This is particularly useful for applications that need to know the call path.

For example, VDN 25201 has VDN Override feature on. VDN 25201 can either route the call to VDN 25202 or VDN 25204. VDN Override is not administered on 25202 and 25204, based on conditions set up at the vector associated with VDN 25201. Both VDN 25202 and 25204 route the call to VDN 25203. Then VDN 25203 routes the call to an agent. If VDN 25201 and the agent's station are both monitored, but not VDN 25202 and 25204, the agent's station monitoring can tell from the distributingDevice whether the path of a call involves 24202 or 24204 when 25201 is called. Also note that, in the Delivered and Established events for the agent's station monitoring, the calledDevice will be 25201 and the lastRedirectionDevice will also be 25201 (if VDN 25203 is monitored, the lastRedirectionDevice will change to 25203).

⇒ NOTE:

Proper switch administration of the VDN Override feature is required on the G3 switch in order to receive a useful distributingDevice. The distributingDevice contains the originally called device if such administration is not performed on the G3 switch.

- ucid*** [optional] Specifies the Universal Call ID (UCID) of the resulting newCall. The UCID is a unique call identifier across switches and the network. A valid UCID is a null-terminated ASCII character string. If there is no UCID associated with this call, the *ucid* contains the ATT_NULL_UCID (a 20-character string of all zeros). This parameter is supported by private data version 5 and later only.
- callOriginatorInfo*** [optional] Specifies the *callOriginatorType* of the call originator such as coin call, 800-service call, or cellular call. This information is from the network, not from the DEFINITY switch. The type is defined in the Bell Communications Research (Bellcore) publication, "Local Exchange Routing Guide," (document number TR-EOP-000085). A list of the currently defined codes (June 1994) is in the Detailed Information sub-section of the "Delivered Event" section in this chapter. This parameter is supported by private data version 5 and later only.
- flexibleBilling*** [optional] Specifies whether the Flexible Billing feature is allowed for this call and the Flexible Billing customer option is assigned on the switch. If this parameter is set to TRUE, the billing rate can be changed for the incoming 900-type call using the Set Bill Rate Service. This parameter is supported by private data version 5 and later only.

Detailed Information:

In addition to the information given below, see the "Event Report Detailed Information" section in this chapter.

- **Distributing Device** — There was no support for the *distributingDevice* parameter in G3PD before Release 2.2. In Release 2.1, the *calledDevice* always contains the originally called device and the *distributingDevice*, if it is different from the *calledDevice*, is not reported.

In Release 1, the *calledDevice* contains the originally called device if there is no *distributingDevice* or contains the *distributingDevice* if call vectoring with VDN override feature of the PBX is turned on. In the later case, the originally called device is not reported.

- **Last Redirection Device** — There is only limited support for this parameter. An application must understand the limitations of this parameter in order to use the information correctly.

⇒ NOTE:

The accuracy of the information provided in this parameter depends on how an application monitors the devices involved in a call scenario. Experimentation may be required before an application can use this information.

⇒ NOTE:

This parameter provides the last device known by the G3PD through monitor services that redirects the call or diverts the call to the device (alertingDevice, answeringDevice, queued) to which the call arrives. The redirection device can be a VDN, ACD Split, or station device. The following call scenarios describe this parameter and its limitations.

Call Scenario 1:

- Both caller and agent device are monitored.
- Caller dials an ACD Split (not monitored) or a VDN (not monitored) to connect to the agent.
- Call arrives at the agent station.
 - If the caller dials the ACD Split directly, the Delivered/Established Events sent to both caller and the agent will have the ACD Split as the lastRedirectionDevice.

⇒ NOTE:

If the caller calls the VDN, instead of the ACD Split, and the VDN sends the call to the ACD Split, the Delivered/Established Events sent to both the caller and the agent will have the VDN as the lastRedirectionDevice. The last redirection device in the PBX is actually the ACD Split.

⇒ NOTE:

If the caller dials the VDN, the VDN sends the call to the ACD Split, and the call is queued at the ACD Split before the agent receives the call, the Delivered/Established Events will have the VDN as the lastRedirectionDevice. The last redirection device in the PBX is actually the ACD Split.

⇒ NOTE:

If the caller calls from an external device, the agent station receives the same lastRedirectionDevice information.

Call Scenario 2:

- Both caller and agent device are monitored.
- Caller dials an ACD Split (not monitored) or a VDN (monitored) to connect to the agent.
- Call arrives at the agent station.

Same results as in the call scenario 1, except in the following case.

- If the caller dials the VDN, the VDN sends the call to the ACD Split, and the call is queued at the ACD Split before the agent receives the call, the Queued Event will have the VDN as the lastRedirectionDevice. The Delivered/Established Events will have the ACD Split as the lastRedirectionDevice.
- If the caller calls from an external device, the agent station receives the same lastRedirectionDevice information.

Call Scenario 3:

- Both caller and the answering party are monitored.
- Caller dials a number (having no effect on the result whether it is monitored or not) and call goes to the first coverage point (not monitored).
- Call goes to the second coverage point (answering station).
- Call arrives at the answering station.
 - The Delivered Event sent to the caller will have the dialed number as the lastRedirectionDevice when call arrives at the first coverage point.
 - The Delivered/Established Events sent to both caller and the answering party will have the first coverage point as the lastRedirectionDevices when call arrives at the answering party.

Call Scenario 4:

- Caller is not monitored, but answering party is monitored.
- Caller dials a number (having no effect on the result whether it is monitored or not) and call goes to the first coverage point (not monitored).
- Call goes to the second coverage point (answering station).
- Call arrives at the answering station.

⇒ NOTE:

The Delivered/Established Events sent to the answering party will have the dialed number as the lastRedirectionDevice event though the first coverage point redirects the call to the answering party.

Call Scenario 5:

- Caller is not monitored, but answering party is monitored.
- Caller dials a number (having no effect on the result whether it is monitored or not) and call goes to the first coverage point (monitored).
- Call goes to the second coverage point (answering station).
- Call arrives at the answering station.
 - The Delivered Event sent to the first coverage point will have the dialed number as the lastRedirectionDevice.

- The Delivered/Established Events sent to the answering party will have the first coverage point as the lastRedirectionDevice.
- The trunkGroup (private data version 5) trunk (private data versions 2-4), split, lookaheadInfo, userEnteredCode, and userInfo private parameters contain the most recent information about a call, while the originalCallInfo contains the original values for this information. If the most recent values are the same as the original values, the original values are not repeated in the originalCallInfo.
- How to Collect userEnteredCode (UEC)
- The following are steps for setting up VDNs, simple vector steps and CSTA Monitor Service requests required for a client application to receive UECs from the switch.

1. Administer a VDN and a vector on the G3 switch with a collect digits step and route command to a second VDN. See “Call Scenario 1:” and “Call Scenario 2:”.

The purpose of this VDN is to collect UEC, but it will not report the UEC to the G3PD, even if the VDN is monitored. The route command must redirect the call to a second VDN. The first VDN doesn't have to be monitored by any client application.

2. Administer a second VDN and vector to receive the redirected call from the first VDN.

The purpose of this second VDN is to report the UEC to the G3PD. Thus it must be monitored by a cstaMonitorCallsViaDevice service request from at least one client. This VDN should redirect the call to its destination. The destination can be a station extension, an ACD split, or another VDN.

If the destination is a station extension and if the station is monitored by a cstaMonitorDevice service request, the station monitor will receive the UEC collected by the first VDN.

If the destination is an ACD split and if an agent station in the split is monitored by a cstaMonitorDevice service request, the station monitor will receive the UEC collected by the first VDN.

If the destination is a VDN and if the VDN is monitored by a cstaMonitorCallsViaDevice Service request, the VDN monitor will not receive the UEC collected by the first VDN.


UEC is reported in Delivered Event Reports (for detailed information, see “Call Scenario 1:” and “Call Scenario 2:”). If multiple UECs are collected by multiple VDNs in call processing, only the most recently collected UEC is reported.

Limitations

- A monitored VDN only reports the UEC it receives (UEC collected in a previous VDN). It will not report UEC it collects or UEC collected after the call is redirected from the VDN.
- A station monitor reports only the UEC that is received by the VDN that redirects the call to the station, provided that the VDN is monitored (see “Call Scenario 2:”).

Call Scenario 1:

- If VDN 24101 is mapped to vector 1, vector 1 has the following steps:
 1. Collect 16 digits after announcement extension 1000
 2. Route to 24102
 3. Stop
- If VDN 24102 is mapped to vector 2, vector 2 has the following steps:
 1. Route to 24103
 2. Stop
- If 24103 is a station extension, the following can occur:
 - When a call is arrived on VDN 24101, the caller will hear the announcement and the switch will wait for the caller to enter 16 digits. After the 16 digits are collected in time (if the collect digits step is timed out, the next step is executed), the call is routed to VDN 24102. The VDN 24102 routes the call to station 24103.
 - If VDN 24101 is monitored using `cstaMonitorCallsViaDevice`, the User Entered Digits will NOT be reported in the Delivered Event Report (Call Delivered to an ACD Device) for the VDN 24101 monitor. This is because the Delivered Event Report is sent before the digits are collected.
 - If VDN 24102 is monitored using `cstaMonitorCallsViaDevice`, the 16 digits collected by VDN 24101 will be reported in the Delivered Event Report (Call Delivered to an ACD Device) for the VDN 24102 monitor. VDN 24101 monitoring is not required for the VDN 24102 monitor to receive UEC collected by VDN 24101.
 - If VDN 24102 is monitored using `cstaMonitorCallsViaDevice` from any client and station 24103 is monitored using `cstaMonitorDevice`, the 16 digits collected by VDN 24101 will be reported in the Delivered Event Report (Call Delivered to a Station Device) sent to the station 24103 monitor. If the client application is interested in the events reported by the station 24103 monitor only, call filters can be used in the `cstaMonitorCallsViaDevice` service to filter out all event reports from VDN 24102. This will not affect the UEC sent to the station 24103 monitor.

 **NOTE:**

VDN 24102 monitoring (with or without call filters) is required for the station 24103 monitor to receive UEC collected by VDN 24101.

Call Scenario 2:

- If VDN 24201 is mapped to vector 11, vector 11 has the following steps:
 1. Collect 10 digits after announcement extension 2000.
 2. Route to 24202.
 3. Stop.
- If VDN 24202 is mapped to vector 12, vector 12 has the following steps:
 1. Collect 16 digits after announcement extension 3000.
 2. Route to 24203.
 3. Stop.
- If VDN 24203 is mapped to vector 13, vector 13 has the following steps:
 1. Queue to main split 2 priority.
 2. Stop.

Where split 2 is a vector-controlled ACD split that has agent extensions 24301, 24302, 24303.

When a call arrives on VDN 24201, the caller will hear an announcement and the switch will wait for the caller to enter 10 digits. After the 10 digits are collected in time, the call is routed to VDN 24202. When the call arrives on VDN 24202, the caller will hear an announcement and the switch will wait for the caller to enter 16 digits. After the 16 digits are collected in time, the call is routed to VDN 24203. The VDN 24203 queues the call to ACD Split 2. If the agent at station 24301 is available, the call is sent to station 24301.

If VDN 24201 is monitored using `cstaMonitorCallsViaDevice`, the 10 digits collected by VDN 24201 will not be reported in the Delivered Event Report (Call Delivered to an ACD Device) sent for the VDN 24201 monitor. This occurs because the Delivered Event Report is sent before the digits are collected.

If VDN 24202 is monitored using `cstaMonitorCallsViaDevice`, the 10 digits collected by VDN 24201 will be reported in the Delivered Event Report (Call Delivered to an ACD Device) sent for the VDN 24202 monitor.

If VDN 24203 is monitored using `cstaMonitorCallsViaDevice`, the 16 digits collected by VDN 24202 will be reported in the Delivered Event Report (Call Delivered to an ACD Device) sent for the VDN 24203 monitor. However, the 10 digits collected by VDN 24201 will not be reported in the Delivered Event for the VDN 24203 monitor.

The `cstaMonitorCallsViaDevice` service receives only the most recent UEC.

If VDN 24202 and VDN 24203 are both monitored using `cstaMonitorCallsViaDevice` from any client, and station 24301 is monitored using `cstaMonitorDevice`, only the 16 digits collected by VDN 24202 will be reported in the Delivered Event Report (Call Delivered to a Station Device) for the station 24301 monitor. The `cstaMonitorDevice` service will receive the UEC that is received by the VDN that redirects calls to the station.

⇒ NOTE:

In order to receive the UEC for station monitoring, the VDN that receives the UEC and redirects calls to the station must be monitored. For example, if VDN 24203 is not monitored by any client, a `cstaMonitorDevice` Service on station 24301 will not receive the 16 digits collected by VDN 24202.

Call Originator Type - The type is defined in a Bell Communications Research (Bellcore) publication, "Local Exchange Routing Guide," (document number TR-EOP-000085). A list of defined codes, as of June 1994, is shown in Table 9-1:

Table 9-1. Call Originator Type

Code	Description
00	Identified Line — No Special Treatment
01	Multiparty — ANI Cannot Be Provided
02	ANI Failure
06	Hotel/Motel — DN Not Accompanied by Automatic Room ID
07	Special Operator Handling Required
20	AIOD — Listed DN of PBX Sent
23	Coin or Non-Coin — Line Status Unknown
24	800 Service Call
27	Coin Call
29	Prison/Inmate Service
30 - 32	Intercept
34	Telco Operator Handled Call
40 - 49	Locally Determined By Carrier
52	Out WATS
60	Telecommunication Relay Service (TRS) — Station Paid
61	Type 1 Cellular
62	Type 2 Cellular
63	Roamer Cellular

Table 9-1. Call Originator Type

Code	Description
66	TRS — From Hotel/Motel
67	TRS — From Restricted Line
70	Private Pay Station
93	Private Virtual Network Call

⇒ NOTE:

Although each value in callOriginatorType has a special meaning, neither the G3 PBX nor the G3PD interprets these values. The values in callOriginatorType are from the network and the application should interpret the meaning of a particular value based on Bellcore's specification.

Syntax

```
#include <acs.h>
#include <csta.h>

// CSTADeliveredEvent

typedef struct
{
    ACSHandle_t      acsHandle;
    EventClass_t    eventClass;    // CSTAUNSOLICITED
    EventType_t     eventType;    // CSTA_DELIVERED
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            CSTAMonitorCrossRefID_t monitorCrossRefId;
            union
            {
                CSTADeliveredEvent_t delivered;
            } u;
        } cstaUnsolicited;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTADeliveredEvent_t
{
    ConnectionID_t      connection;
    SubjectDeviceID_t   alertingDevice;
    CallingDeviceID_t   callingDevice;
    CalledDeviceID_t    calledDevice;
    RedirectionDeviceID_t lastRedirectionDevice;
    LocalConnectionState_t localConnectionInfo;
    CSTAEventCause_t    cause;
} CSTADeliveredEvent_t;
```

Private Data Version 6 Syntax

If private data accompanies a CSTADeliveredEvent, then the private data would be stored in the location that the application specified as the privateData parameter in the acsGetEventBlock() or acsGetEventPoll() request. If the privateData pointer is set to NULL in these requests, then the CSTADeliveredEvent does not deliver private data to the application. If acsGetEventBlock() or acsGetEventPoll() returns a Private Data length of 0, then no private data is provided with this event.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTDeliveredEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_t          eventType;// ATT_DELIVERED
    union
    {
        ATTDeliveredEvent_t deliveredEvent;
    } u;
} ATTEvent_t;

typedef struct ATTDeliveredEvent_t
{
    ATTDeliveredType_t      deliveredType;
    DeviceID_t              trunkGroup;
    DeviceID_t              trunkMember;
    DeviceID_t              split;
    ATTLookaheadInfo_t      lookaheadInfo;
    ATTUserEnteredCode_t    userEnteredCode;
    ATTUserToUserInfo_t     userInfo;
    ATTReasonCode_t         reason;
    ATTOriginalCallInfo_t   originalCallInfo;
    CalledDeviceID_t        distributingDevice;
    ATTUCID_t               ucid;
    ATTCallOriginatorType_t callOriginatorInfo;
    Boolean                  flexibleBilling;
} ATTDeliveredEvent_t;

typedef enum ATTDeliveredType_t
{
    DELIVERED_TO_ACD          = 1,
    DELIVERED_TO_STATION     = 2,
    DELIVERED_OTHER          = 3 // not in use
} ATTDeliveredType_t;
```

Private Data Version 6 Syntax (Continued)

```
typedef struct ATTLookaheadInfo_t
{
    ATTInterflow_t      type;
    ATTPriority_t       priority;
    short               hours;
    short               minutes;
    short               seconds;
    DeviceID_t          sourceVDN;
    ATTUnicodeDeviceID_t sourceVDN; // sourceVDN in Unicode
} ATTLookaheadInfo_t;

typedef enum ATTInterflow_t
{
    LAI_NO_INTERFLOW   = -1,    // indicates info not present
    LAI_ALL_INTERFLOW  = 0,
    LAI_THRESHOLD_INTERFLOW= 1,
    LAI_VECTORIZING_INTERFLOW= 2
} ATTInterflow_t;

typedef enum ATTPriority_t
{
    LAI_NOT_IN_QUEUE   = 0,
    LAI_LOW             = 1,
    LAI_MEDIUM         = 2,
    LAI_HIGH            = 3,
    LAI_TOP             = 4
} ATTPriority_t;

typedef struct ATTUnicodeDeviceID_t
{
    short               count;
    unsigned shortvalue[64];
} ATTUnicodeDeviceID_t;

typedef struct ATTUserEnteredCode_t
{
    ATTUserEnteredCodeType_t type;
    ATTUserEnteredCodeIndicator_t indicator;
    char                 data[ATT_MAX_USER_CODE];
    DeviceID_t           collectVDN;
} ATTUserEnteredCode_t;

typedef enum ATTUserEnteredCodeType_t
{
    UE_NONE             = -1,    // indicates not specified
    UE_ANY              = 0,
    UE_LOGIN_DIGITS     = 2,
    UE_CALL_PROMPTER   = 5,
    UE_DATA_BASE_PROVIDED= 17,
    UE_TONE_DETECTOR   = 32
} ATTUserEnteredCodeType_t;
```

Private Data Version 6 Syntax (Continued)

```

typedef enum ATTUserEnteredCodeIndicator_t
{
    UE_COLLECT= 0,
    UE_ENTERED= 1
} ATTUserEnteredCodeIndicator_t;

typedef struct ATTUserToUserInfo_t
{
    ATTUUIProtocolType_ttype;
    struct {
        short      length;      // 0 indicates UUI not present
        unsigned   char      value[ATT_MAX_USER_INFO];
    } data;
} ATTUserToUserInfo_t;

typedef enum ATTUUIProtocolType_t
{
    UUI_NONE          = -1,    // indicates not specified
    UUI_USER_SPECIFIC= 0,    // user-specific
    UUI_IA5_ASCII     = 4      // null terminated ascii
// character string
} ATTUUIProtocolType_t;

typedef enum ATReasonCode_t
{
    AR_NONE          = 0, // no reason code specified
    AR_ANSWER_NORMAL = 1, // answer supervision from
                        // the network or internal answer
    AR_ANSWER_TIMED  = 2, // assumed answer based on
                        // internal timer
    AR_ANSWER_VOICE_ENERGY = 3, // voice energy detection by
                        // classifier
    AR_ANSWER_MACHINE_DETECTED= 4, // answering machine detected
    AR_SIT_REORDER   = 5, // switch equipment congestion
    AR_SIT_NO_CIRCUIT = 6, // no circuit or channel
                        // available
    AR_SIT_INTERCEPT = 7, // number changed
    AR_SIT_VACANT_CODE = 8, // unassigned number
    AR_SIT_INEFFECTIVE_OTHER= 9, // invalid number
    AR_SIT_UNKNOWN    = 10, // normal unspecified
    AR_IN_QUEUE       = 11, // call still in queue - for
                        // Delivered Event only
    AR_SERVICE_OBSERVER= 12 // service observer connected
} ATReasonCode_t

```

Private Data Version 6 Syntax (Continued)

```
typedef struct ATTOriginalCallInfo_t
{
    ATTReasonForCallInfo_t reason;
    CallingDeviceID_t    callingDevice;
    CalledDeviceID_t    calledDevice;
    DeviceID_t          trunkGroup;
    DeviceID_t          trunkMember;
    ATTLookaheadInfo_t  lookaheadInfo;
    ATTUserEnteredCode_t userEnteredCode;
    ATTUserToUserInfo_t userInfo;
    ATTUCID_t           ucid;
    ATTCallOriginatorType_t callOriginatorInfo;
    Boolean              flexibleBilling;
} ATTOriginalCallInfo_t;

typedef enum ATTReasonForCallInfo_t
{
    OR_NONE          = 0, // indicates Original
                      // Call Info not present
    OR_CONSULTATION = 1,
    OR_CONFERENCED  = 2,
    OR_TRANSFERRED  = 3,
    OR_NEW_CALL     = 4
} ATTReasonForCallInfo_t;

typedef ExtendedDeviceID_t CallingDeviceID_t;

typedef ExtendedDeviceID_t CalledDeviceID_t;

typedef char ATTUCID_t[64];

typedef struct ATTCallOriginatorInfo_t
{
    Boolean    hasInfo;    // If FALSE, no
                          // callOriginatorType
    short     callOriginatorType;
} ATTCallOriginatorInfo_t;
```

Private Data Version 5 Syntax

If private data accompanies a CSTADeliveredEvent, then the private data would be stored in the location that the application specified as the privateData parameter in the acsGetEventBlock() or acsGetEventPoll() request. If the privateData pointer is set to NULL in these requests, then the CSTADeliveredEvent does not deliver private data to the application. If acsGetEventBlock() or acsGetEventPoll() returns a Private Data length of 0, then no private data is provided with this event.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV5DeliveredEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_t          eventType;// ATT_DELIVERED
    union
    {
        ATTV5DeliveredEvent_t deliveredEvent;
    } u;
} ATTEvent_t;

typedef struct ATTV5DeliveredEvent_t
{
    ATTDeliveredType_t      deliveredType;
    DeviceID_t              trunkGroup;
    DeviceID_t              trunkMember;
    DeviceID_t              split;
    ATTLookaheadInfo_t      lookaheadInfo;
    ATTUserEnteredCode_t    userEnteredCode;
    ATTV5UserToUserInfo_t   userInfo;
    ATTReasonCode_t         reason;
    ATTV5OriginalCallInfo_t originalCallInfo;
    CalledDeviceID_t        distributingDevice;
    ATTUCID_t               ucid;
    ATTCallOriginatorType_t callOriginatorInfo;
    Boolean                  flexibleBilling;
} ATTV5DeliveredEvent_t;

typedef enum ATTDeliveredType_t
{
    DELIVERED_TO_ACD          = 1,
    DELIVERED_TO_STATION     = 2,
    DELIVERED_OTHER          = 3 // not in use
} ATTDeliveredType_t;
```

Private Data Version 5 Syntax (Continued)

```

typedef struct ATTLookaheadInfo_t
{
    ATTInterflow_t      type;
    ATTPriority_t       priority;
    short               hours;
    short               minutes;
    short               seconds;
    DeviceID_t          sourceVDN;
    ATTUnicodeDeviceID_t sourceVDN; // sourceVDN in Unicode
} ATTLookaheadInfo_t;

typedef enum ATTInterflow_t
{
    LAI_NO_INTERFLOW    = -1,    // indicates info not present
    LAI_ALL_INTERFLOW   = 0,
    LAI_THRESHOLD_INTERFLOW= 1,
    LAI_VECTORIZING_INTERFLOW= 2
} ATTInterflow_t;

typedef enum ATTPriority_t
{
    LAI_NOT_IN_QUEUE    = 0,
    LAI_LOW              = 1,
    LAI_MEDIUM          = 2,
    LAI_HIGH            = 3,
    LAI_TOP              = 4
} ATTPriority_t;

typedef struct ATTUnicodeDeviceID_t
{
    short               count;
    unsigned shortvalue[64];
} ATTUnicodeDeviceID_t;

typedef struct ATTUserEnteredCode_t
{
    ATTUserEnteredCodeType_t type;
    ATTUserEnteredCodeIndicator_t indicator;
    char                 data[ATT_MAX_USER_CODE];
    DeviceID_t           collectVDN;
} ATTUserEnteredCode_t;

typedef enum ATTUserEnteredCodeType_t
{
    UE_NONE              = -1,    // indicates not specified
    UE_ANY               = 0,
    UE_LOGIN_DIGITS     = 2,
    UE_CALL_PROMPTER    = 5,
    UE_DATA_BASE_PROVIDED= 17,
    UE_TONE_DETECTOR    = 32
} ATTUserEnteredCodeType_t;

```


Private Data Version 5 Syntax (Continued)

```
typedef enum ATTUserEnteredCodeIndicator_t
{
    UE_COLLECT= 0,
    UE_ENTERED= 1
} ATTUserEnteredCodeIndicator_t;

typedef struct ATTV5UserToUserInfo_t
{
    ATTUUIProtocolType_ttype;
    struct {
        short      length;    // 0 indicates UII not present
        unsigned   char      value[33];
    } data;
} ATTV5UserToUserInfo_t;

typedef enum ATTUUIProtocolType_t
{
    UII_NONE          = -1,    // indicates not specified
    UII_USER_SPECIFIC= 0,    // user-specific
    UII_IA5_ASCII     = 4      // null terminated ascii
// character string
} ATTUUIProtocolType_t;

typedef enum ATTReasonCode_t
{
    AR_NONE          = 0, // no reason code specified
    AR_ANSWER_NORMAL = 1, // answer supervision from
                        // the network or internal answer
    AR_ANSWER_TIMED  = 2, // assumed answer based on
                        // internal timer
    AR_ANSWER_VOICE_ENERGY = 3, // voice energy detection by
                        // classifier
    AR_ANSWER_MACHINE_DETECTED= 4, // answering machine detected
    AR_SIT_REORDER   = 5, // switch equipment congestion
    AR_SIT_NO_CIRCUIT = 6, // no circuit or channel
                        // available
    AR_SIT_INTERCEPT = 7, // number changed
    AR_SIT_VACANT_CODE = 8, // unassigned number
    AR_SIT_INEFFECTIVE_OTHER= 9, // invalid number
    AR_SIT_UNKNOWN    = 10, // normal unspecified
    AR_IN_QUEUE       = 11, // call still in queue - for
                        // Delivered Event only
    AR_SERVICE_OBSERVER= 12 // service observer connected
} ATTReasonCode_t
```

Private Data Version 5 Syntax (Continued)

```
typedef struct ATTV5OriginalCallInfo_t
{
    ATTReasonForCallInfo_t reason;
    CallingDeviceID_t    callingDevice;
    CalledDeviceID_t    calledDevice;
    DeviceID_t          trunkGroup;
    DeviceID_t          trunkMember;
    ATTLookaheadInfo_t  lookaheadInfo;
    ATTUserEnteredCode_t userEnteredCode;
    ATTV5UserToUserInfo_t userInfo;
    ATTUCID_t           ucid;
    ATTCallOriginatorType_t callOriginatorInfo;
    Boolean              flexibleBilling;
} ATTV5OriginalCallInfo_t;

typedef enum ATTReasonForCallInfo_t
{
    OR_NONE          = 0, // indicates Original
                      // Call Info not present
    OR_CONSULTATION = 1,
    OR_CONFERENCED  = 2,
    OR_TRANSFERRED  = 3,
    OR_NEW_CALL     = 4
} ATTReasonForCallInfo_t;

typedef ExtendedDeviceID_t CallingDeviceID_t;

typedef ExtendedDeviceID_t CalledDeviceID_t;

typedef char ATTUCID_t[64];

typedef struct ATTV5CallOriginatorInfo_t
{
    Boolean    hasInfo;    // If FALSE, no
                          // callOriginatorType
    short     callOriginatorType;
} ATTV5CallOriginatorInfo_t;
```

Private Data Version 4 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV4DeliveredEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_t eventType; // ATTV4_DELIVERED
    union
    {
        ATTV4DeliveredEvent_t tv4deliveredEvent;
    } u;
} ATTEvent_t;

typedef struct ATTV4DeliveredEvent_t
{
    ATTDeliveredType_t deliveredType;
    DeviceID_t trunk;
    DeviceID_t trunkMember;
    DeviceID_t split;
    ATTV4LookaheadInfo_t lookaheadInfo;
    ATTUserEnteredCode_t userEnteredCode;
    ATTV5UserToUserInfo_t userInfo;
    ATTReasonCode_t reason;
    ATTV4OriginalCallInfo_t originalCallInfo;
    CalledDeviceID_t distributingDevice;
} ATTV4DeliveredEvent_t;

typedef enum ATTDeliveredType_t
{
    DELIVERED_TO_ACD = 1,
    DELIVERED_TO_STATION = 2,
    DELIVERED_OTHER = 3 // not in use
} ATTDeliveredType_t;

typedef struct ATTV4LookaheadInfo_t {
    ATTInterflow_t type;
    ATTPriority_t priority;
    short hours;
    short minutes;
    short seconds;
    DeviceID_t sourceVDN;
} ATTV4LookaheadInfo_t;
```

Private Data Version 4 Syntax (Continued)

```
typedef enum ATTInterflow_t
{
    LAI_NO_INTERFLOW= -1, // indicates info not present
    LAI_ALL_INTERFLOW= 0,
    LAI_THRESHOLD_INTERFLOW= 1,
    LAI_VECTORIZING_INTERFLOW= 2
} ATTInterflow_t;

typedef enum ATTPriority_t
{
    LAI_NOT_IN_QUEUE= 0,
    LAI_LOW          = 1,
    LAI_MEDIUM       = 2,
    LAI_HIGH         = 3,
    LAI_TOP          = 4
} ATTPriority_t;

typedef struct ATTUserEnteredCode_t
{
    ATTUserEnteredCodeType_t    type;
    ATTUserEnteredCodeIndicator_t indicator;
    char                        data[ATT_MAX_USER_CODE];
    DeviceID_t                  collectVDN;
} ATTUserEnteredCode_t;

typedef enum ATTUserEnteredCodeType_t
{
    UE_NONE          = -1, // indicates not specified
    UE_ANY           = 0,
    UE_LOGIN_DIGITS= 2,
    UE_CALL_PROMPTER= 5,
    UE_DATA_BASE_PROVIDED= 17,
    UE_TONE_DETECTOR= 32
} ATTUserEnteredCodeType_t;

typedef enum ATTUserEnteredCodeIndicator_t
{
    UE_COLLECT= 0,
    UE_ENTERED= 1
} ATTUserEnteredCodeIndicator_t;

typedef struct ATTV5UserToUserInfo_t
{
    ATTUUIProtocolType_t type;
    struct {
        short length; // 0 indicates UUI not present
        unsigned char value[33];
    } data;
} ATTV5UserToUserInfo_t;
```

Private Data Version 4 Syntax (Continued)

```

typedef enum ATTUIProtocolType_t
{
    UUI_NONE          = -1,    // indicates not specified
    UUI_USER_SPECIFIC = 0,    // user-specific
    UUI_IA5_ASCII     = 4,    // null terminated ascii
// character string
} ATTUIProtocolType_t;

typedef enum ATTReasonCode_t
{
    AR_NONE          = 0, // no reason code specified
    AR_ANSWER_NORMAL = 1, // answer supervision from
                        // the network or internal answer
    AR_ANSWER_TIMED  = 2, // assumed answer based on
                        // internal timer
    AR_ANSWER_VOICE_ENERGY= 3, // voice energy detection by
                        // classifier
    AR_ANSWER_MACHINE_DETECTED= 4, // answering machine detected
    AR_SIT_REORDER   = 5,    // switch equipment
                        // congestion
    AR_SIT_NO_CIRCUIT = 6,    // no circuit or channel
                        // available
    AR_SIT_INTERCEPT = 7,    // number changed
    AR_SIT_VACANT_CODE = 8,    // unassigned number
    AR_SIT_INEFFECTIVE_OTHER = 9, // invalid number
    AR_SIT_UNKNOWN     = 10,   // normal unspecified
    AR_IN_QUEUE        = 11 // call still in queue - for
// Delivered Event only} ATTReasonCode_t

typedef struct ATTV4OriginalCallInfo_t
{
    ATTReasonForCallInfo_t reason;
    CallingDeviceID_t      callingDevice;
    CalledDeviceID_t       calledDevice;
    DeviceID_t             trunk;
    DeviceID_t             trunkMember;
    ATTV4LookaheadInfo_t  lookaheadInfo;
    ATTUserEnteredCode_t  userEnteredCode;
    ATTV5UserToUserInfo_t userInfo;
} ATTV4OriginalCallInfo_t;

typedef enum ATTReasonForCallInfo_t
{
    OR_NONE          = 0, // indicates Original
                        // Call Info not present
    OR_CONSULTATION  = 1,
    OR_CONFERENCED   = 2,
    OR_TRANSFERRED   = 3,
    OR_NEW_CALL      = 4
} ATTReasonForCallInfo_t;

typedef ExtendedDeviceID_tCallingDeviceID_t;

typedef ExtendedDeviceID_tCalledDeviceID_t;

```

Private Data Versions 2 and 3 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV3DeliveredEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_t eventType; // ATTV3_DELIVERED
    union
    {
        ATTV3DeliveredEvent_t tv3deliveredEvent;
    } u;
} ATTEvent_t;

typedef struct ATTV3DeliveredEvent_t
{
    ATTDeliveredType_t    deliveredType;
    DeviceID_t            trunk;
    DeviceID_t            trunkMember;
    DeviceID_t            split;
    ATTV4LookaheadInfo_t lookaheadInfo;
    ATTUserEnteredCode_t userEnteredCode;
    ATTV5UserToUserInfo_t userInfo;
    ATTReasonCode_t      reason;
    ATTV4OriginalCallInfo_t originalCallInfo;
} ATTV3DeliveredEvent_t;

typedef enum ATTDeliveredType_t
{
    DELIVERED_TO_ACD      = 1,
    DELIVERED_TO_STATION = 2,
    DELIVERED_OTHER      = 3 // not in use
} ATTDeliveredType_t;

typedef struct ATTV4LookaheadInfo_t
{
    ATTInterflow_t    type;
    ATTPriority_t     priority;
    short             hours;
    short             minutes;
    short             seconds;
    DeviceID_t        sourceVDN;
} ATTV4LookaheadInfo_t;

typedef enum ATTInterflow_t
{
    LAI_NO_INTERFLOW= -1, // indicates info not present
    LAI_ALL_INTERFLOW= 0,
    LAI_THRESHOLD_INTERFLOW= 1,
    LAI_VECTORING_INTERFLOW= 2
} ATTInterflow_t;
```

Private Data Versions 2 and 3 Syntax (Continued)

```

typedef enum ATTPriority_t
{
    LAI_NOT_IN_QUEUE= 0,
    LAI_LOW      = 1,
    LAI_MEDIUM   = 2,
    LAI_HIGH     = 3,
    LAI_TOP      = 4
} ATTPriority_t;

typedef struct ATTUserEnteredCode_t
{
    ATTUserEnteredCodeType_t    type;
    ATTUserEnteredCodeIndicator_tindicator;
    char                        data[ATT_MAX_USER_CODE];
    DeviceID_t                  collectVDN;
} ATTUserEnteredCode_t;

typedef enum ATTUserEnteredCodeType_t
{
    UE_NONE          = -1,    // indicates not specified
    UE_ANY           = 0,
    UE_LOGIN_DIGITS= 2,
    UE_CALL_PROMPTER= 5,
    UE_DATA_BASE_PROVIDED= 17,
    UE_TONE_DETECTOR= 32
} ATTUserEnteredCodeType_t;

typedef enum ATTUserEnteredCodeIndicator_t
{
    UE_COLLECT= 0,
    UE_ENTERED= 1
} ATTUserEnteredCodeIndicator_t;

typedef struct ATTV5UserToUserInfo_t
{
    ATTUUIProtocolType_ttype;
    struct {
        short          length;    // 0 indicates UUI not present
        unsigned charvalue[33];
    } data;
} ATTV5UserToUserInfo_t;

typedef enum ATTUUIProtocolType_t
{
    UUI_NONE          = -1,    // indicates not specified
    UUI_USER_SPECIFIC= 0,    // user-specific
    UUI_IA5_ASCII     = 4      // null terminated ascii
                                // character string
} ATTUUIProtocolType_t;

```

Private Data Versions 2 and 3 Syntax (Continued)

```
typedef enum ATReasonCode_t
{
AR_NONE                = 0, // no reason code specified
AR_ANSWER_NORMAL       = 1, // answer supervision from
                        // the network or internal answer
AR_ANSWER_TIMED        = 2, // assumed answer based on
                        // internal timer
AR_ANSWER_VOICE_ENERGY= 3, // voice energy detection by
                        // classifier
AR_ANSWER_MACHINE_DETECTED= 4, // answering machine detected
AR_SIT_REORDER         = 5,  // switch equipment
                        // congestion
AR_SIT_NO_CIRCUIT      = 6,  // no circuit or channel
                        // available
AR_SIT_INTERCEPT     = 7,  // number changed
AR_SIT_VACANT_CODE     = 8,  // unassigned number
AR_SIT_INEFFECTIVE_OTHER = 9, // invalid number
AR_SIT_UNKNOWN         = 10, // normal unspecified
} ATReasonCode_t

typedef struct ATTV4OriginalCallInfo_t
{
    ATReasonForCallInfo_treason;
    CallingDeviceID_t    callingDevice;// original info
    CalledDeviceID_t     calledDevice;// original info
    DeviceID_t           trunk; // original info
    DeviceID_t           trunkMember;// not in use
    ATTV4LookaheadInfo_t lookaheadInfo; // original info
    ATTUserEnteredCode_t userEnteredCode;// original info
    ATTV5UserToUserInfo_tuserInfo;// original info
} ATTV4OriginalCallInfo_t;

typedef enum ATReasonForCallInfo_t
{
    OR_NONE            = 0,    // indicates Original
                        // Call Info not present
    OR_CONSULTATION    = 1,
    OR_CONFERENCED     = 2,
    OR_TRANSFERRED     = 3,
    OR_NEW_CALL        = 4
} ATReasonForCallInfo_t;

typedef ExtendedDeviceID_tCallingDeviceID_t;

typedef ExtendedDeviceID_tCalledDeviceID_t;
```


Diverted Event

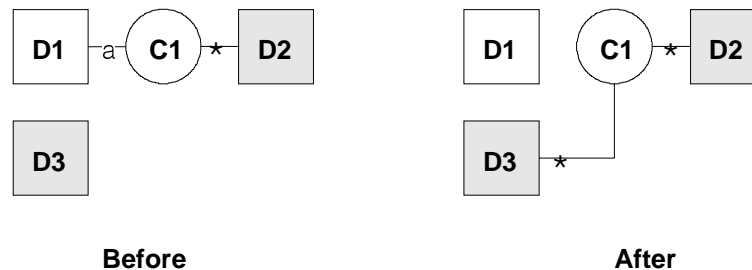
Direction: Switch to Client

Event: CSTADivertedEvent

Service Parameters: monitorCrossRefID, connection, divertingDevice, newDestination, localConnectionInfo, cause

Functional Description:

The Diverted Event Report indicates a call that has been deflected or diverted from a monitored device, and is no longer present at the device.



The Diverted Event Report is sent to notify the client application that event reports for a call will no longer be provided. This event report is sent under the following circumstances:

- When a call enters a new VDN or ACD split that is being monitored.¹ For example, if a call leaves one monitored ACD device and enters another, a Call Diverted Event Report is sent to the monitor for the first ACD device. A Delivered Event Report must have been received by the ACD monitoring before the Diverted Event Report.
- When a call leaves a monitored station, without having been dropped or disconnected, this report is sent to the monitor for the station. A Delivered Event Report must have been received by the station monitoring before the Diverted Event Report.
- When a call that had been alerting at the station leaves the station because:
 - One member of a coverage and/or answer group answers a call offered to a coverage group. In this case, all other members of the coverage and/or answer group that were alerting for the call receive a Diverted Event Report.
 - A call has gone to AUDIX coverage and the Coverage Response Interval (CRI) has elapsed (the principal's call is redirected).

1. Described in the "Delivered Event" section.

- The principal answers the call while the coverage point is alerting and the coverage point is dropped from the call.
- For stations that are members of a TEG group with no associated TEG button (typically analog stations).
- The monitored station is an analog phone and an alerting call is now alerting elsewhere (gone to coverage) because:
 - The pick-up feature is used to answer a call alerting an analog principal's station.
 - An analog phone call is sent to coverage due to "no answer" (the analog station's call is redirected).

This event report will not be sent if the station is never alerted or if it retains a simulated bridge appearance until the call is dropped/disconnected. Examples of situations when this event is not sent are:

- Bridging
- Call forwarding
- Calls to a TEG (multifunction set with TEG button)
- Cover-All
- Coverage/Busy
- Incoming PCOL calls (multifunction sets)
- Pick-up for multifunction set principals

This event report will never follow an Established Event Report and is always preceded by a Delivered Event Report.

⇒ NOTE:

Event reporting has been changed for Release 3.10 and later. Prior to Release 3.10, the Diverted Event was only sent to the device monitor from which the call was diverted. Therefore, a monitored calling device could not tell whether the call was diverted from a called device (the calling device will receive only a Delivered Event when the call is alerting at a coverage point) or the called device was still on the call when the call went to the coverage point. Only the called device could tell if the call was diverted. All other devices or calls, if monitored, did not receive the Diverted Event.

⇒ NOTE:

This has been changed since Release 3.10 and applies to streams opened with Private Data Version 5 only. If an application opens a stream with Private Data Version 4 or earlier, it will not be affected by this change. When the application opens a Private Data Version 5 stream, the Diverted Event is sent for all station device monitors, ACD devices (VDNs and ACD Splits) monitors, and call monitors independent of whether the diverting device is the monitored device. A station device monitor, an ACD device monitor, or a call monitor will be reported whether a call is leaving or staying at a previously alerted device (when a call goes to a coverage point) via the presence or absence of the Diverted Event, respectively. Note that this change only affects the Diverted event reporting; there is no private data change for the Diverted Event itself.

Service Parameters:

<i>monitorCrossRefID</i>	[mandatory] Contains the handle to the monitor request for which this event is reported.
<i>connection</i>	[mandatory] Specifies the connection that was alerting.
<i>divertingDevice</i>	[optional — partially supported] Specifies the device from which the call was diverted.
<i>newDestination</i>	[optional — partially supported] Specifies the device to which the call was diverted.
<i>localConnectionInfo</i>	[optional — supported] Specifies the local connection state as perceived by the monitored device on this call. This is provided for the <i>cstaMonitorDevice</i> requests only. A value of <i>CS_NONE</i> indicates that the local connection state is unknown.
<i>cause</i>	[optional — supported] Specifies the cause for this event. The following cause is supported: <ul style="list-style-type: none"> ■ <i>EC_REDIRECTED</i> — The call has been redirected.

Detailed Information:

See the “Event Report Detailed Information” section in this chapter.

Syntax

```
#include <acs.h>
#include <csta.h>

// CSTADivertedEvent

typedef struct
{
    ACSHandle_tacsHandle;
    EventClass_teventClass; // CSTAUNSOLICITED
    EventType_teventType;   // CSTA_DIVERTED
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_teventHeader;
    union
    {
        struct
        {
            CSTAMonitorCrossRefID_tmonitorCrossRefId;
            union
            {
                CSTADivertedEvent_t diverted;
            } u;
        } cstaUnsolicited;
    } event;
    charheap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTADivertedEvent_t
{
    ConnectionID_t      connection;
    SubjectDeviceID_t  divertingDevice;
    CalledDeviceID_t   newDestination;
    LocalConnectionState_t localConnectionInfo;
    CSTAEventCause_t   cause;
} CSTADivertedEvent_t;

typedef ExtendedDeviceID_tSubjectDeviceID_t;

typedef ExtendedDeviceID_tCalledDeviceID_t;
```

Entered Digits Event (Private)

Direction: Switch to Client

Event: CSTAPrivateStatusEvent

Private Data Event: ATTEnteredDigitsEvent

Service Parameters: monitorCrossRefID

Private Parameters: connection, digits, localConnectionInfo, cause

Functional Description:

The Entered Digits Event is sent when a DTMF tone detector attached to a call and DTMF tones are received. The tone detector is disconnected when the far end answers or "#" is detected. The digits reported include: 0-9, "*", and "#". The digit string includes the "#", if present. Up to 24 digits can be entered.

Service Parameters:

monitorCrossRefID [mandatory] Contains the handle to the monitor request for which this event is reported.

Private Parameters:

connection [mandatory] Specifies the callID of the call for which this event is reported.

digits [mandatory] Specifies the digits user entered. The digits reported include: 0-9, "*", and "#". The digit string includes the "#", if present. The digit string is null terminated.

localConnectionInfo [optional] Specifies the local connection state as perceived by the monitored device on this call. A value of CS_NONE is always specified.

cause [optional] Specifies the cause for this event.

Detailed Information:

See the "Event Report Detailed Information" section in this chapter.

Syntax

```
#include <acs.h>
#include <csta.h>

// CSTAPrivateStatusEvent

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass;    // CSTAUNSOLICITED
    EventType_t eventType;    // CSTA_PRIVATE_STATUS
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            CSTAMonitorCrossRefID_t monitorCrossRefId;
            union
            {
                CSTAPrivateEvent_t privateStatus;
            } u;
        } cstaUnsolicited;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;
```

Private Parameter Syntax

If private data accompanies a CSTAPrivateStatusEvent, then the private data would be stored in the location that the application specified as the privateData parameter in the acsGetEventBlock() or acsGetEventPoll() request. If the privateData pointer is set to NULL in these requests, then the CSTAPrivateStatusEvent does not deliver private data to the application. If acsGetEventBlock() or acsGetEventPoll() returns a Private Data length of 0, then no private data is provided with this event.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTEnteredDigitsEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_teventType;// ATT_ENTERED_DIGITS
    union
    {
        ATTEnteredDigitsEvent_tenteredDigitsEvent;
    } u;
} ATTEvent_t;

// ATT Entered Digits Structure
typedef struct ATTEnteredDigitsEvent_t
{
    ConnectionID_t      connection;
    char                digits[ATT_MAX_ENTERED_DIGITS];
    LocalConnectionState_tlocalConnectionInfo;
    CSTAEventCause_t   cause;
} ATTEnteredDigitsEvent_t;
```

Established Event

Direction: Switch to Client

Event: CSTAEstablishedEvent

Private Data Event: ATTEstablishedEvent (private data version 6), ATTV5EstablishedEvent (private data version 5), ATTV4EstablishedEvent (private data version 4), ATTV3EstablishedEvent (private data versions 2 and 3)

Service Parameters: monitorCrossRefID, establishedConnection, answeringDevice, callingDevice, calledDevice, lastRedirectionDevice, localConnectionInfo, cause

Private Parameters: trunkGroup, trunkMember, split, lookaheadInfo, userEnteredCode, userInfo, reason, originalCallInfo, distributingDevice, ucid, callOriginatorInfo, flexibleBilling

Functional Description:

The Established Event Report indicates that the switch detects that a device has answer or connected to a call.



The Established Event Report is sent as follows:

- When a cstaMakePredictiveCall call is delivered to an on-PBX party (after having been answered at the destination) and the on-PBX party answers the call (picked up handset or cut-through after zip tone).
- When a cstaMakePredictiveCall call is placed to an off-PBX destination and an ISDN CONNect message is received from an ISDN-PRI facility.
- When a cstaMakePredictiveCall call is placed to an off-PBX destination and the call classifier detects an answer or a Special Information Tone (SIT) administered to answer.
- When a call is delivered to an on-PBX party and the on-PBX party has answered the call (picked up handset or cut-through after zip tone).
- When a call is redirected to an off-PBX destination, and the ISDN CONN (ISDN connect) message is received from an ISDN-PRI facility.
- Any time a station is connected to a call (picked up on a bridged call appearance, service observing, busy verification, etc.).

In general, the Established Event Report is not sent for split or vector announcements nor it is sent for the attendant group (0).

Multiple Established Event Reports

Multiple Established Event Reports may be sent for a specific call. For example, when a call is first picked up by coverage, the event is sent to the active monitors for the coverage party, as well as to the active monitors for all other extensions already on the call. If the call is then bridged onto by the principal, the Established Event Report is then sent to the monitors for the principal, as well as to the monitors for all other extensions active on the call.

Multiple Established Event Reports may also be sent for the same extension on a call. For example, when a call is first picked up by a member of a bridge, TEG, PCOL, an Established Event Report is generated. If that member goes on-hook and then off-hook again while another member of the particular group is connected on the call, a second Established Event Report will be sent for the same extension. This event report is not sent for split or vector announcements, nor it is sent for the attendant group (0).

Service Parameters:

<i>monitorCrossRefID</i>	[mandatory] Contains the handle to the monitor request for which this event is reported.
<i>establishedConnection</i>	[mandatory] Specifies the endpoint that joined the call.
<i>answeringDevice</i>	[mandatory] Specifies the device that joined the call. <ul style="list-style-type: none"> ■ For outgoing calls over PRI facilities¹ —“connected number” from the ISDN CONN (ISDN connect) message. ■ If the device being connected is on-PBX, then the extension of the device is specified (primary extension for TEGs, PCOLs, bridging).
<i>callingDevice</i>	[mandatory] Specifies the calling device. The following rules apply: <ul style="list-style-type: none"> ■ For internal calls originated at an on-PBX station — the station’s extension is specified. ■ For outgoing calls over PRI facilities —“calling number” from the ISDN SETUP message or its assigned trunk identifier is specified, if the “calling number” does not exist (it is NULL). ■ For incoming calls over PRI facilities —“calling number” from the ISDN SETUP message or its assigned trunk identifier is specified, if the “calling number” does not exist (it is NULL). ■ For incoming calls over non-PRI facilities — the calling party number is generally not available. The assigned trunk identifier² is provided instead. ■ The trunk group number is specified only when the calling party number is not available. ■ For calls originated at a bridged call appearance —the principal’s extension is specified.
<i>calledDevice</i>	[mandatory — partially supported] Specifies the originally called device. The following rules apply: <ul style="list-style-type: none"> ■ For outgoing calls over PRI facilities — “called number” from the ISDN SETUP message is specified. If the “called number” does not exist (it is NULL), the deviceIDStatus is ID_NOT_KNOWN. ■ For incoming calls over PRI facilities — “called number” from the ISDN SETUP message is specified. If the “called number” does not exist (it is NULL), the deviceIDStatus is ID_NOT_KNOWN.

- For incoming calls over non-PRI facilities — the principal extension is specified. It may be a group extension for TEG, hunt group, VDN. If the switch is administered to modify the DNIS digits, then the modified DNIS string is specified.
- For incoming calls to PCOL, the deviceID is ID_NOT_KNOWN.
- For incoming calls to a TEG (principal) group, the TEG group extension is specified.
- For incoming calls to a principal with bridges, the principal's extension is specified.
- If the called device is on-PBX and the call did not come over a PRI facility, the extension of the party dialed is specified.

<i>lastRedirectionDevice</i>	[optional — limited support] Specifies the previously redirection/alerted device in the case where the call was redirected/diverted to the answeringDevice.
<i>localConnectionInfo</i>	[optional — supported] Specifies the local connection state as perceived by the monitored device on this call. This is provided for cstaMonitorDevice requests only. A value of CS_NONE indicates that the local connection state is unknown.
<i>cause</i>	[optional — supported] Specifies the cause for this event. The following causes are supported: <ul style="list-style-type: none"> ■ EC_TRANSFER — A call transfer has occurred. This cause has higher precedence than the following two. See Blind Transfer in the “Detailed Information:” section. ■ EC_KEY_CONFERENCE — Indicates that the event report occurred at a bridged device. This cause has higher precedence than the following one ■ EC_NEW_CALL — The call has not yet been transferred. ■ EC_PARK — The call is connected due to picking up a parked call.

- EC_ACTIVE_MONITOR — This is the cause value if the Established Event Report resulted from a Single Step Conference request and the Single Step Conference request is for PT_ACTIVE. For details, see “Single Step Conference Call Service” in Chapter 4.
- EC_SILENT_MONITOR —
 1. This is the cause value if the Established Event Report resulted from a Single Step Conference request and the Single Step Conference request is for PT_SILENT. For details, see “Single Step Conference Call Service” in Chapter 4.
 2. This is also the cause value if the Established Event Report resulted from a Service Observer (with either listen-only or listen-and-talk mode) joining the call. In this case, the reason parameter in private data version 5 and later will have AR_SERVICE_OBSERVER. Private data version 4 and earlier will not have this information.



NOTE:

An application cannot distinguish between case 1 from and case 2 using the cause value only. However, the reason parameter in private data version 5 and later indicates whether the EC_SILENT_MONITOR is from Single Step Conference or Service Observer. The EC_SILENT_MONITOR for AR_SERVICE_OBSERVER is a G3V6 feature.

-
1. For outgoing calls over non_PRI facilities, there is no Established Event Report. A Network Reached Event Report is generated instead.
 2. The trunk identifier is a dynamic identifier and it cannot be used to access a trunk in the G3 switch.

Private Parameters:

<i>trunkGroup</i>	[optional] Specifies the trunk group number from which the call originated. Beginning with G3V8, trunk group number is provided regardless of whether the callingDevice is available. Prior to G3V8, trunk group number is provided only if the callingDevice is unavailable. This parameter is supported by private data version 5 and later only.
<i>trunk</i>	[optional] Specifies the trunk group number from which the call originated. Trunk group number is provided only if the callingDevice is unavailable. This parameter is supported by private data versions 2, 3, and 4 only.
<i>trunkMember</i>	[optional — limited supported] This parameter is supported beginning with G3V4. It specifies the trunk member number from which the call originated. Beginning with G3V8, trunk member number is provided regardless of whether the callingDevice is available. Prior to G3V8, trunk member number is provided only if the callingDevice is unavailable.
<i>split</i>	[optional] Specifies the ACD split extension to which the call is delivered.
<i>distributingDevice</i>	[optional] Specifies the ACD or VDN device that distributed the call to the station. This information is provided only when the call was processed by the switch ACD or Call Vectoring processing and is only sent for a station monitor (i.e., the delivery type is DELIVERED_TO_STATION). This parameter is supported by private data version 4 and later.

⇒ NOTE:

The calledDevice specifies the originally called device. In most ACD call scenarios, calledDevice and distributingDevice have the same device ID. However, in call scenarios that involve call vectoring with the VDN Override feature turned on, calledDevice and distributingDevice may have different deviceIDs. Incoming calls that arrived at the same calledDevice may be distributed to an agent via different call paths that have more than one VDN involved. If the VDN Override feature is used on the calledDevice, the distributingDevice specifies the VDN that distributed the call to the agent. This is particularly useful for applications that need to know the call path.

⇒ NOTE:

Proper switch administration is required on the G3 switch in order to receive a useful distributingDevice. The distributingDevice contains the originally called device if such administration is not performed on the G3 switch.

lookaheadInfo [optional] Specifies the lookahead interflow information received from the established call. The lookahead interflow is a G3 switch feature that routes some of the incoming calls from one switch to another so that they can be handled more efficiently and will not be lost. The lookahead interflow information is provided by the switch that overflows the call. A routing application may use the lookahead interflow information to determine the destination of the call. See the G3 Feature Description for more information about lookahead interflow. If the lookahead interflow type is set to "LAI_NO_INTERFLOW", no lookahead interflow private data is provided with this event.

userEnteredCode [optional] Specifies the code/digits that may have been entered by the caller through the G3 call prompting feature or the collected digits feature. If the userEnteredCode code is set to "UE_NONE", no userEnteredCode private data is provided with this event.

userInfo [optional] Contains user-to-user information. This parameter allows an application to associate caller information, up to 32 or 96 bytes, with a call. This information may be a customer number, credit card number, alphanumeric digits, or a binary string.

Prior to G3V8, the maximum length of userInfo was 32 bytes. Beginning with G3V8, the maximum length of userInfo is increased to 96 bytes.

⇒ NOTE:

An application using private data version 5 and earlier can only receive a maximum of 32-byte data for userInfo, regardless of the size data that is sent by the switch.

The following UUI protocol types are supported:

- UUI_NONE — There is no data provided in the data parameter.
- UUI_USER_SPECIFIC — The content of the data parameter is a binary string. The correct size (maximum of 32 or 96 bytes) of data must be specified in the size parameter.
- UUI_IA5_ASCII — The content of the data parameter must be a null-terminated IA5 (ASCII) character string. The correct size (maximum of 32 or 96 bytes excluding the null terminator) of data must be specified in the size parameter.

reason [optional] Specifies the reason that caused this event. The following reasons are supported:

Issue 1 — December 2001

- AR_NONE— indicate no value specified for reason.
- AR_ANSWER_NORMAL— answer supervision from the network or internal answer.
- AR_ANSWER_TIMED — assumed answer based on internal timer.
- AR_ANSWER_VOICE_ENERGY — voice energy detection from a call classifier.
- AR_ANSWER_MACHINE_DETECTED — answering machine detected
- AR_SIT_REORDER — switch equipment congestion
- AR_SIT_NO_CIRCUIT — no circuit or channel available
- AR_SIT_INTERCEPT — number changed
- AR_SIT_VACANT_CODE — unassigned number
- AR_SIT_INEFFECTIVE_OTHER — invalid number
- AR_SIT_UNKNOWN — normal unspecified

originalCallInfo

[optional] Specifies the original call information. Note that information is not repeated in the `originalCallInfo`, if it is already reported in the CSTA service parameters or in the private data. For example, the `callingDevice` and `calledDevice` in the `originalCallInfo` will be NULL, if the `callingDevice` and the `calledDevice` in the CSTA service parameters are the original calling and called devices. Only when the original devices are different from the most recent `callingDevice` and `calledDevice`, the `callingDevice` and `calledDevice` in the `originalCallInfo` will be set. If the `userEnteredCode` in the private data is the original (first time entered) `userEnteredCode`, the `userEnteredCode` in the `originalCallInfo` will be UE_NONE. Only when new (second time entered) `userEnteredCode` is received, will `originalCallInfo` have the original `userEnteredCode`.

⇒ NOTE:

For the Established Event sent to the `newCall` of a Consultation Call, the `originalCallInfo` is taken from the `activeCall` specified in the Consultation Call request. Thus the application can pass the original call information between two calls. The `calledDevice` of the Consultation Call must reside on the same switch and must be monitored via the same Tserver.

The `originalCallInfo` includes the original call information received by the `activeCall` in the Consultation Call request. The original call information includes:

- **reason** — the reason for the originalCallInfo. The following reasons are supported.
 - OR_NONE — no originalCallInfo provided
 - OR_CONFERENCED — call conferenced
 - OR_CONSULTATION — consultation call
 - OR_TRANSFERRED — call transferred
 - OR_NEW_CALL — new call
- **callingDevice** — the original callingDevice received by the activeCall.
- **calledDevice** — the original calledDevice received by the activeCall.
- **trunk** — the original trunk group received by the activeCall. This parameter is supported by private data versions 2, 3, and 4.
- **trunkGroup** — the original trunkGroup received by the activeCall. This parameter is supported by private data version 5 and later only.
- **trunkMember** (G3V4 switches and later) — the original trunkMember received by the activeCall.
- **lookaheadInfo** — the original lookaheadInfo received by the activeCall.
- **userEnteredCode** — the original userEnteredCode received by the activeCall.
- **userInfo** — the original userInfo received by the activeCall.

Prior to G3V8, the maximum length of userInfo was 32 bytes. Beginning with G3V8, the maximum length of userInfo is increased to 96 bytes.

 **NOTE:**

An application using private data version 5 and earlier can only receive a maximum of 32-byte data for userInfo, regardless of the size data that is sent by the switch.

- **ucid** — the original ucid of the call. This parameter is supported by private data version 5 and later only.
- **callOriginatorInfo** — the original callOriginatorInfo for the call. This parameter is supported by private data version 5 and later only.

- **flexibleBilling** — the original flexibleBilling information of the call. This parameter is supported by private data version 5 and later only.

ucid [optional] Specifies the Universal Call ID (UCID) of the call. The UCID is a unique call identifier across switches and the network. A valid UCID is a null-terminated ASCII character string. If there is no UCID associated with this call, the `ucid` contains the `ATT_NULL_UCID` (a 20-character string of all zeros). This parameter is supported by private data version 5 and later only.

callOriginatorInfo [optional] Specifies the `callOriginatorType` of the call originator such as coin call, 800-service call, or cellular call. This information is from the network, not from the DEFINITY switch. The type is defined in the Bellcore publication, "Local Exchange Routing Guide," (document number TR-EOP-000085). A list of the currently defined codes (June 1994) is in the Detailed Information sub-section of the "Delivered Event" section in this chapter. This parameter is supported by private data version 5 and later only.

flexibleBilling [optional] Specifies whether the Flexible Billing feature is allowed for this call and the Flexible Billing customer option is assigned on the switch. If this parameter is set to `TRUE`, the billing rate can be changed for the incoming 900-type call using the Set Bill Rate Service. This parameter is supported by private data version 5 and later only.

Detailed Information:

See the "Event Report Detailed Information" section in this chapter.

- **Call Classification** — For `cstaMakePredictiveCall`, the switch uses the Call Classification process, along with a variety of internal and external events, to determine a predictive (switch-classified call) call outcome. Whenever the called endpoint is external, a call classifier is used.

The classifier is inserted in the connection as soon as the digits have been outpulsed (sent out on a circuit). A call is classified as either answered (Established Event) or dropped (Call Cleared/Connection Cleared Event).

A Delivered Event is reported to the application, but it is not the final classification. "Non-classified energy" is always treated as an answer classification and reported to the application in an Established Event. A modem answer back tone results in a Call Cleared/Connection Cleared Event. Special Information Tone (SIT) detection is reported to the application as an Established Event or a Call Cleared/Connection Cleared Event, depending on the customer's administration preference. Answer Machine Detection (AMD) is reported as an Established Event or a Call Cleared/Connection Cleared Event, depending on administration or call options.

- **Last Redirection Device** — There is only limited support for this parameter. An application must understand the limitations of this parameter in order to use the information correctly.
- **Blind Transfer** — Application designers using caller information to pop screens should refer to “Transferring or Conferencing a Call Together with Screen Pop Information” in Chapter 3 that describes how to coordinate the passing of caller information across applications.

An EC_TRANSFER in the cause indicates that a blind transfer occurred before the call was established. A blind transfer is a call transfer operation that completes before the receiving party answers. Thus, when the receiving party answers, the caller and the receiving party are connected. The transferring party is not part of the connection. In terms of manual operations, it is as if the transferring party presses the transfer button to put the caller on hold, dials the receiving party, and immediately presses the transfer button again (while the call is ringing at the receiving party). Since the transfer occurs between the time the call rings at the receiving party (CSTA Delivered Event) and the time that the receiving party answers the call (CSTA Established Event), the callingDevice changes between these two events.

⇒ NOTE:

The G3 PBX will not send a Transferred Event for the blind transfer operation to the receiving party before or after the Established event. An application must look in the CSTA Established Event for the callingDevice (ANI) information.

- **Consultation Transfer** — (Also known as “manual transfer” or “supervised transfer”) — The transfer does not complete before the receiving party answers. Specifically, the transferring party and the receiving party are connected and can consult before the transfer occurs. The caller is not connected to this consultation conversation. In terms of manual operations, it is as if the transferring party presses the transfer button to put the caller on hold, dials the receiving party, the receiving party answers, the transferring and receiving parties consult, and then the transferring party presses transfer again to transfer the call. Since the transfer occurs after the time that the receiving party answers the consultation call (after the CSTA Established Event), there is no EC_TRANSFER in the cause of the Established Event.

⇒ NOTE:

ANI screen pop applications should follow the guidelines in Chapter 3. ANI screen pop in cases where the user does a consultation transfer manually from the telephone requires information that appears on a cstaMonitorDevice of the transferring party. If both the transferring party and the receiving party run applications that use the same G3PD, then this requirement is met. To do an ANI screen pop in this case, an application must look in the CSTA Transfer Event for the ANI information. An ANI screen pop for a manual consultation

transfer is done in this way at the time the call transfers, not when the consultation call rings or is answered.

Basic information on the design of application pop screens using caller information is found in Chapter 3. Additional details and interactions are found in the “Event Report Detailed Information” section in this chapter. The notes above are special cases and do not reflect the recommended design.

The trunkGroup, trunk, split, lookaheadInfo, userEnteredCode, userInfo private parameters contain the most recent information about a call, while the originalCallInfo contains the original values for this information. If the most recent values are the same as the original values, the original values are not repeated in the originalCallInfo.

Syntax

```
#include <acs.h>
#include <csta.h>

// CSTAEstablishedEvent

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTAUNSOLICITED
    EventType_t eventType; // CSTA_ESTABLISHED
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            CSTAMonitorCrossRefID_t monitorCrossRefId;
            union
            {
                CSTAEstablishedEvent_t established;
            } u;
        } cstaUnsolicited;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAEstablishedEvent_t
{
    ConnectionID_t          establishedConnection;
    SubjectDeviceID_t      answeringDevice;
    CallingDeviceID_t      callingDevice;
    CalledDeviceID_t       calledDevice;
    RedirectionDeviceID_t  lastRedirectionDevice;
    LocalConnectionState_t localConnectionInfo;
    CSTAEventCause_t       cause;
} CSTAEstablishedEvent_t;
```

Private Data Version 6 Syntax

If private data accompanies a CSTAEstablishedEvent, then the private data would be stored in the location that the application specified as the privateData parameter in the acsGetEventBlock() or acsGetEventPoll() request. If the privateData pointer is set to NULL in these requests, then the CSTAEstablishedEvent does not deliver private data to the application. If acsGetEventBlock() or acsGetEventPoll() returns a Private Data length of 0, then no private data is provided with this event.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTEstablishedEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_t eventType; // ATT_ESTABLISHED
    union
    {
        ATTEstablishedEvent_t establishedEvent;
    } u;
} ATTEvent_t;

typedef struct ATTEstablishedEvent_t
{
    DeviceID_t          trunkGroup; // most recent info
    DeviceID_t          trunkMember; // not in use
    DeviceID_t          split;      // for monitor device
                                // (station) only
    ATTLookaheadInfo_t lookaheadInfo; // most recent info
    ATTUserEnteredCode_t userEnteredCode; // most recent info
    ATTUserToUserInfo_t userInfo; // most recent info
    ATTReasonCode_t     reason;
    ATTOriginalCallInfo_t originalCallInfo; // original info
    CalledDeviceID_t    distributingDevice; // most recent
info
    ATTUCID_t           ucid;
    ATTCallOriginatorInfo_t callOriginatorInfo;
    Boolean              flexibleBilling;
} ATTEstablishedEvent_t;

typedef struct ATTLookaheadInfo_t
{
    ATTInterflow_t      type;
    ATTPriority_t       priority;
    short               hours;
    short               minutes;
    short               seconds;
    DeviceID_t          sourceVDN;
    ATTUnicodeDeviceID_t uSourceVDN; // sourceVDN in Unicode
} ATTLookaheadInfo_t;
```

Private Data Version 6 Syntax (Continued)

```
typedef struct ATTUnicodeDeviceID_t
{
    short          count;
    unsigned short value[64];
} ATTUnicodeDeviceID_t;

typedef enum ATTInterflow_t
{
LAI_NO_INTERFLOW= -1, // indicates Info not present
    LAI_ALL_INTERFLOW      = 0,
    LAI_THRESHOLD_INTERFLOW = 1,
    LAI_VECTORIZING_INTERFLOW = 2
} ATTInterflow_t;

typedef enum ATTPriority_t
{
    LAI_NOT_IN_QUEUE      = 0,
    LAI_LOW                = 1,
    LAI_MEDIUM            = 2,
    LAI_HIGH               = 3,
    LAI_TOP                = 4
} ATTPriority_t;

typedef struct ATTUserEnteredCode_t
{
    ATTUserEnteredCodeType_t    type;
    ATTUserEnteredCodeIndicator_t indicator;
    char                        data[ATT_MAX_USER_CODE];
    DeviceID_t                  collectVDN; // VDN that reports
                                     // this userEnteredCode
} ATTUserEnteredCode_t;

typedef enum ATTUserEnteredCodeType_t
{
    UE_NONE          = -1, // indicates not specified
    UE_ANY           = 0,
    UE_LOGIN_DIGITS = 2,
    UE_CALL_PROMPTER= 5,
    UE_DATA_BASE_PROVIDED= 17,
    UE_TONE_DETECTOR= 32
} ATTUserEnteredCodeType_t;

typedef enum ATTUserEnteredCodeIndicator_t
{
    UE_COLLECT= 0,
    UE_ENTERED= 1
} ATTUserEnteredCodeIndicator_t;
```

Private Data Version 6 Syntax (Continued)

```
typedef struct ATTUserToUserInfo_t
{
    ATTUUIProtocolType_t type;
    struct {
        short          length; // 0 - UUI not present
        unsigned char  value[ATT_MAX_USER_INFO];
    } data;
} ATTUserToUserInfo_t;

typedef enum ATTUUIProtocolType_t
{
    UUI_NONE           = -1, // indicates not specified
    UUI_USER_SPECIFIC = 0, // user-specific
    UUI_IA5_ASCII      = 4 // null terminated ascii
                        // character string
} ATTUUIProtocolType_t;

typedef enum ATTReasonCode_t
{
    AR_NONE           = 0, // no reason code specified
    AR_ANSWER_NORMAL = 1, // answer supervision from
                        // the network or internal
                        // answer
    AR_ANSWER_TIMED = 2, // assumed answer based on
                        // internal timer
    AR_ANSWER_VOICE_ENERGY = 3, // voice energy detection
                        // by classifier
    AR_ANSWER_MACHINE_DETECTED = 4, // answering machine
                        // detected
    AR_SIT_REORDER = 5, // switch equipment
                        // congestion
    AR_SIT_NO_CIRCUIT = 6, // no circuit or channel
                        // available
    AR_SIT_INTERCEPT = 7, // number changed
    AR_SIT_VACANT_CODE = 8, // unassigned number
    AR_SIT_INEFFECTIVE_OTHER = 9, // invalid number
    AR_SIT_UNKNOWN = 10, // normal unspecified
    AR_IN_QUEUE = 11, // call still in queue - for
                        // Delivered Event only
    AR_SERVICE_OBSERVER = 12 // service observer
                        // connected
} ATTReasonCode_t
```

Private Data Version 6 Syntax (Continued)

```
typedef struct ATTOriginalCallInfo_t
{
    ATTReasonForCallInfo_t  reason;
    CallingDeviceID_t       callingDevice;
    CalledDeviceID_t        calledDevice;
    DeviceID_t              trunkGroup;
    DeviceID_t              trunkMember;
    ATTLookaheadInfo_t      lookaheadInfo;
    ATTUserEnteredCode_t    userEnteredCode;
    ATTUserToUserInfo_t     userInfo;
    ATTUCID_t               ucid;
    ATTCallOriginatorInfo_t callOriginatorInfo;
    Boolean                  flexibleBilling;
} ATTOriginalCallInfo_t;

typedef enum ATTReasonForCallInfo_t
{
    OR_NONE           = 0, // indicates info not present
    OR_CONSULTATION  = 1,
    OR_CONFERENCED   = 2,
    OR_TRANSFERRED   = 3,
    OR_NEW_CALL      = 4
} ATTReasonForCallInfo_t;

typedef ExtendedDeviceID_t CallingDeviceID_t;

typedef ExtendedDeviceID_t CalledDeviceID_t;

typedef char ATTUCID_t[64];

typedef struct ATTCallOriginatorInfo_t
{
    Boolean    hasInfo; // if FALSE, no callOriginatorType
    short     callOriginatorType;
} ATTCallOriginatorInfo_t;
```


Private Data Version 5 Syntax

If private data accompanies a CSTAEstablishedEvent, then the private data would be stored in the location that the application specified as the privateData parameter in the acsGetEventBlock() or acsGetEventPoll() request. If the privateData pointer is set to NULL in these requests, then the CSTAEstablishedEvent does not deliver private data to the application. If acsGetEventBlock() or acsGetEventPoll() returns a Private Data length of 0, then no private data is provided with this event.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV5EstablishedEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_t eventType; // ATT_ESTABLISHED
    union
    {
        ATTV5EstablishedEvent_t establishedEvent;
    } u;
} ATTEvent_t;

typedef struct ATTV5EstablishedEvent_t
{
    DeviceID_t      trunkGroup; // most recent info
    DeviceID_t      trunkMember; // not in use
    DeviceID_t      split;      // for monitor device
                                // (station) only
    ATTLookaheadInfo_t lookaheadInfo; // most recent info
    ATTUserEnteredCode_t userEnteredCode; // most recent info
    ATTV5UserToUserInfo_t userInfo; // most recent info
    ATTReasonCode_t reason;
    ATTV5OriginalCallInfo_t originalCallInfo; // original info
    CalledDeviceID_t distributingDevice; // most recent
info
    ATTUCID_t      ucid;
    ATTCallOriginatorInfo_t callOriginatorInfo;
    Boolean        flexibleBilling;
} ATTV5EstablishedEvent_t;

typedef struct ATTLookaheadInfo_t
{
    ATTInterflow_t      type;
    ATTPriority_t       priority;
    short               hours;
    short               minutes;
    short               seconds;
    DeviceID_t          sourceVDN;
    ATTUnicodeDeviceID_t uSourceVDN; // sourceVDN in Unicode
} ATTLookaheadInfo_t;
```

Private Data Version 5 Syntax (Continued)

```
typedef struct ATTUnicodeDeviceID_t
{
    short          count;
    unsigned short value[64];
} ATTUnicodeDeviceID_t;

typedef enum ATTInterflow_t
{
LAI_NO_INTERFLOW= -1, // indicates Info not present
    LAI_ALL_INTERFLOW      = 0,
    LAI_THRESHOLD_INTERFLOW = 1,
    LAI_VECTORIZING_INTERFLOW = 2
} ATTInterflow_t;

typedef enum ATTPriority_t
{
    LAI_NOT_IN_QUEUE      = 0,
    LAI_LOW                = 1,
    LAI_MEDIUM            = 2,
    LAI_HIGH               = 3,
    LAI_TOP                = 4
} ATTPriority_t;

typedef struct ATTUserEnteredCode_t
{
    ATTUserEnteredCodeType_t    type;
    ATTUserEnteredCodeIndicator_t indicator;
    char                        data[ATT_MAX_USER_CODE];
    DeviceID_t                  collectVDN; // VDN that reports
                                     // this userEnteredCode
} ATTUserEnteredCode_t;

typedef enum ATTUserEnteredCodeType_t
{
    UE_NONE      = -1, // indicates not specified
    UE_ANY       = 0,
    UE_LOGIN_DIGITS = 2,
    UE_CALL_PROMPTER= 5,
    UE_DATA_BASE_PROVIDED= 17,
    UE_TONE_DETECTOR= 32
} ATTUserEnteredCodeType_t;

typedef enum ATTUserEnteredCodeIndicator_t
{
    UE_COLLECT= 0,
    UE_ENTERED= 1
} ATTUserEnteredCodeIndicator_t;
```

Private Data Version 5 Syntax (Continued)

```
typedef struct ATTV5UserToUserInfo_t
{
    ATTUUIProtocolType_t type;
    struct {
        short          length; // 0 - UII not present
        unsigned char  value[33];
    } data;
} ATTV5UserToUserInfo_t;

typedef enum ATTUUIProtocolType_t
{
    UII_NONE           = -1, // indicates not specified
    UII_USER_SPECIFIC = 0, // user-specific
    UII_IA5_ASCII      = 4 // null terminated ascii
                        // character string
} ATTUUIProtocolType_t;

typedef enum ATTRReasonCode_t
{
    AR_NONE           = 0, // no reason code specified
    AR_ANSWER_NORMAL = 1, // answer supervision from
                        // the network or internal
                        // answer
    AR_ANSWER_TIMED = 2, // assumed answer based on
                        // internal timer
    AR_ANSWER_VOICE_ENERGY = 3, // voice energy detection
                        // by classifier
    AR_ANSWER_MACHINE_DETECTED = 4, // answering machine
                        // detected
    AR_SIT_REORDER = 5, // switch equipment
                        // congestion
    AR_SIT_NO_CIRCUIT = 6, // no circuit or channel
                        // available
    AR_SIT_INTERCEPT = 7, // number changed
    AR_SIT_VACANT_CODE = 8, // unassigned number
    AR_SIT_INEFFECTIVE_OTHER = 9, // invalid number
    AR_SIT_UNKNOWN = 10, // normal unspecified
    AR_IN_QUEUE = 11, // call still in queue - for
                        // Delivered Event only
    AR_SERVICE_OBSERVER = 12 // service observer
                        // connected
} ATTRReasonCode_t
```

Private Data Version 5 Syntax (Continued)

```
typedef struct ATTV5OriginalCallInfo_t
{
    ATTReasonForCallInfo_t  reason;
    CallingDeviceID_t       callingDevice;
    CalledDeviceID_t        calledDevice;
    DeviceID_t              trunkGroup;
    DeviceID_t              trunkMember;
    ATTLookaheadInfo_t      lookaheadInfo;
    ATTUserEnteredCode_t    userEnteredCode;
    ATTV5UserToUserInfo_t   userInfo;
    ATTUCID_t               ucid;
    ATTV5CallOriginatorInfo_t callOriginatorInfo;
    Boolean                  flexibleBilling;
} ATTV5OriginalCallInfo_t;

typedef enum ATTReasonForCallInfo_t
{
    OR_NONE           = 0, // indicates info not present
    OR_CONSULTATION   = 1,
    OR_CONFERENCED    = 2,
    OR_TRANSFERRED    = 3,
    OR_NEW_CALL       = 4
} ATTReasonForCallInfo_t;

typedef ExtendedDeviceID_t CallingDeviceID_t;

typedef ExtendedDeviceID_t CalledDeviceID_t;

typedef char ATTUCID_t[64];

typedef struct ATTCallOriginatorInfo_t
{
    Boolean    hasInfo; // if FALSE, no callOriginatorType
    short     callOriginatorType;
} ATTCallOriginatorInfo_t;
```

Private Data Version 4 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV4EstablishedEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_t eventType; // ATTV4_ESTABLISHED
    union
    {
        ATTV4EstablishedEvent_t tv4establishedEvent;
    } u;
} ATTEvent_t;

typedef struct ATTV4EstablishedEvent_t
{
    DeviceID_t    trunk;           // most recent info
    DeviceID_t    trunkMember;    // not in use
    DeviceID_t    split;          // for monitor device
                                   // (station) only
    ATTV4LookaheadInfo_t lookaheadInfo; // most recent info
    ATTUserEnteredCode_t userEnteredCode; // most recent info
    ATTV5UserToUserInfo_t userInfo;    // most recent info
    ATTReasonCode_t    reason;
    ATTV4OriginalCallInfo_t originalCallInfo; // original info
    CalledDeviceID_t    distributingDevice; // most recent
                                   // info
} ATTV4EstablishedEvent_t;

typedef struct ATTV4LookaheadInfo_t
{
    ATTInterflow_t    type;
    ATTPriority_t     priority;
    short             hours;
    short             minutes;
    short             seconds;
    DeviceID_t        sourceVDN;
} ATTV4LookaheadInfo_t;
```

Private Data Version 4 Syntax (Continued)

```
typedef enum ATTInterflow_t
{
    LAI_NO_INTERFLOW      = -1, // indicates Info not present
    LAI_ALL_INTERFLOW     = 0,
    LAI_THRESHOLD_INTERFLOW= 1,
    LAI_VECTORIZING_INTERFLOW= 2
} ATTInterflow_t;

typedef enum ATTPriority_t
{
    LAI_NOT_IN_QUEUE= 0,
    LAI_LOW          = 1,
    LAI_MEDIUM       = 2,
    LAI_HIGH         = 3,
    LAI_TOP          = 4
} ATTPriority_t;

typedef struct ATTUserEnteredCode_t
{
    ATTUserEnteredCodeType_t    type;
    ATTUserEnteredCodeIndicator_t indicator;
    char                        data[ATT_MAX_USER_CODE];
    DeviceID_t                  collectVDN; // VDN that
                                   // reports this userEnteredCode
} ATTUserEnteredCode_t;

typedef enum ATTUserEnteredCodeType_t
{
    UE_NONE          = -1, // indicates not specified
    UE_ANY           = 0,
    UE_LOGIN_DIGITS  = 2,
    UE_CALL_PROMPTER = 5,
    UE_DATA_BASE_PROVIDED= 17,
    UE_TONE_DETECTOR= 32
} ATTUserEnteredCodeType_t;

typedef enum ATTUserEnteredCodeIndicator_t
{
    UE_COLLECT      = 0,
    UE_ENTERED      = 1
} ATTUserEnteredCodeIndicator_t;

typedef struct ATTV5UserToUserInfo_t
{
    ATTUUIProtocolType_t type;
    struct {
        short          length; // 0 indicates UUI not
                               // present
        unsigned char  value[33];
    } data;
} ATTV5UserToUserInfo_t;
```

Private Data Version 4 Syntax (Continued)

```
typedef enum ATTUUIProtocolType_t
{
    UUI_NONE          = -1, // indicates not specified
    UUI_USER_SPECIFIC= 0, // user-specific
    UUI_IA5_ASCII= 4   // null terminated ascii
                        // character string
} ATTUUIProtocolType_t;

typedef enum ATTReasonCode_t
{
    AR_NONE          = 0, // no reason code specified
    AR_ANSWER_NORMAL= 1, // answer supervision from
                        // the network or internal
                        // answer
    AR_ANSWER_TIMED= 2, // assumed answer based on
                        // internal timer
    AR_ANSWER_VOICE_ENERGY= 3, // voice energy detection
                        // by classifier
    AR_ANSWER_MACHINE_DETECTED = 4, // answering machine detected
    AR_SIT_REORDER= 5, // switch equipment congestion
    AR_SIT_NO_CIRCUIT= 6, // no circuit or channel
                        // available
    AR_SIT_INTERCEPT= 7, // number changed
    AR_SIT_VACANT_CODE= 8, // unassigned number
    AR_SIT_INEFFECTIVE_OTHER= 9, // invalid number
    AR_SIT_UNKNOWN= 10, // normal unspecified
} ATTReasonCode_t

typedef struct ATTV4OriginalCallInfo_t
{
    ATTReasonForCallInfo_t  reason;
    CallingDeviceID_t       callingDevice;
    CalledDeviceID_t        calledDevice;
    DeviceID_t              trunk;
    DeviceID_t              trunkMember;
    ATTV4LookaheadInfo_t    lookaheadInfo;
    ATTUserEnteredCode_t    userEnteredCode;
    ATTV5UserToUserInfo_t   userInfo;
} ATTV4OriginalCallInfo_t;
```

Private Data Version 4 Syntax (Continued)

```
typedef enum ATTReasonForCallInfo_t
{
    OR_NONE          = 0, // indicates info not present
    OR_CONSULTATION = 1,
    OR_CONFERENCED  = 2,
    OR_TRANSFERRED  = 3,
    OR_NEW_CALL     = 4
} ATTReasonForCallInfo_t;

typedef ExtendedDeviceID_tCallingDeviceID_t;

typedef ExtendedDeviceID_tCalledDeviceID_t;
```


Private Data Versions 2 and 3 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV3EstablishedEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_t eventType; // ATTV3_ESTABLISHED
    union
    {
        ATTV3EstablishedEvent_t tv3establishedEvent;
    } u;
} ATTEvent_t;

typedef struct ATTV3EstablishedEvent_t
{
    DeviceID_t      trunk;           // most recent info
    DeviceID_t      trunkMember;    // not in use
    DeviceID_t      split;          // for monitor device
                                   // (station) only
    ATTV4LookaheadInfo_t lookaheadInfo; // most recent info
    ATTUserEnteredCode_t userEnteredCode; // most recent info
    ATTV5UserToUserInfo_t userInfo;    // most recent info
    ATTReasonCode_t reason;
    ATTV4OriginalCallInfo_t originalCallInfo; // original info
} ATTV3EstablishedEvent_t;

typedef struct ATTV4LookaheadInfo_t
{
    ATTInterflow_t type;
    ATTPriority_t priority;
    short hours;
    short minutes;
    short seconds;
    DeviceID_t sourceVDN;
} ATTV4LookaheadInfo_t;
```

Private Data Versions 2 and 3 Syntax (Continued)

```
typedef enum ATTInterflow_t
{
LAI_NO_INTERFLOW      = -1, // indicates Info not present
    LAI_ALL_INTERFLOW= 0,
    LAI_THRESHOLD_INTERFLOW= 1,
    LAI_VECTORIZING_INTERFLOW= 2
} ATTInterflow_t;

typedef enum ATTPriority_t
{
    LAI_NOT_IN_QUEUE= 0,
    LAI_LOW          = 1,
    LAI_MEDIUM       = 2,
    LAI_HIGH         = 3,
    LAI_TOP          = 4
} ATTPriority_t;

typedef struct ATTUserEnteredCode_t
{
    ATTUserEnteredCodeType_t    type;
    ATTUserEnteredCodeIndicator_t indicator;
    char                        data[ATT_MAX_USER_CODE];
    DeviceID_t                  collectVDN; // VDN that reports
                                   // this userEnteredCode
} ATTUserEnteredCode_t;

typedef enum ATTUserEnteredCodeType_t
{
    UE_NONE          = -1, // indicates not specified
    UE_ANY           = 0,
    UE_LOGIN_DIGITS  = 2,
    UE_CALL_PROMPTER = 5,
    UE_DATA_BASE_PROVIDED= 17,
    UE_TONE_DETECTOR = 32
} ATTUserEnteredCodeType_t;

typedef enum ATTUserEnteredCodeIndicator_t
{
    UE_COLLECT= 0,
    UE_ENTERED= 1
} ATTUserEnteredCodeIndicator_t;

typedef struct ATTV5UserToUserInfo_t
{
    ATTUUIProtocolType_t type;
    struct {
        short          length; // 0 indicates UUI not present
        unsigned charvalue[33];
    } data;
} ATTV5UserToUserInfo_t;
```

Private Data Versions 2 and 3 Syntax (Continued)

```
typedef enum ATTUUIProtocolType_t
{
    UUI_NONE          = -1, // indicates not specified
    UUI_USER_SPECIFIC= 0, // user-specific
    UUI_IA5_ASCII= 4    // null terminated ascii
                        // character string
} ATTUUIProtocolType_t;

typedef enum ATTReasonCode_t
{
    AR_NONE          = 0, // no reason code specified
    AR_ANSWER_NORMAL= 1, // answer supervision from
                        // the network or internal
                        // answer
    AR_ANSWER_TIMED= 2,  // assumed answer based on
                        // internal timer
    AR_ANSWER_VOICE_ENERGY= 3, // voice energy detection
                        // by classifier
    AR_ANSWER_MACHINE_DETECTED = 4, // answering machine
                        // detected
    AR_SIT_REORDER   = 5,  // switch equipment congestion
    AR_SIT_NO_CIRCUIT = 6,  // no circuit or channel
                        // available
    AR_SIT_INTERCEPT = 7, // number changed
    AR_SIT_VACANT_CODE= 8, // unassigned number
    AR_SIT_INEFFECTIVE_OTHER= 9, // invalid number
    AR_SIT_UNKNOWN= 10,    // normal unspecified
} ATTReasonCode_t

typedef struct ATTV4OriginalCallInfo_t
{
    ATTReasonForCallInfo_t    reason;
    CallingDeviceID_t        callingDevice;
    CalledDeviceID_t         calledDevice;
    DeviceID_t               trunk;
    DeviceID_t               trunkMember;
    ATTV4LookaheadInfo_t    lookaheadInfo;
    ATTUserEnteredCode_t     userEnteredCode;
    ATTV5UserToUserInfo_t   userInfo;
} ATTV4OriginalCallInfo_t;
```

Private Data Versions 2 and 3 Syntax (Continued)

```
typedef enum ATTReasonForCallInfo_t
{
    OR_NONE          = 0, // indicates info not present
    OR_CONSULTATION = 1,
    OR_CONFERENCED  = 2,
    OR_TRANSFERRED  = 3,
    OR_NEW_CALL     = 4
} ATTReasonForCallInfo_t;

typedef ExtendedDeviceID_tCallingDeviceID_t;

typedef ExtendedDeviceID_tCalledDeviceID_t;
```

Failed Event

Direction: Switch to Client

Event: CSTAFailedEvent

Service Parameters: monitorCrossRefID, failedConnection, failingDevice, calledDevice, localConnectionInfo, cause

Functional Description:

The Failed Event Report indicates that a call cannot be completed.



This event report is generated when the destination of a call is busy or unavailable, as follows:

- When a call is delivered to an on-PBX station and the station is busy (without coverage and call waiting).
- When a call tries to terminate on an ACD split without going through a vector and the destination ACD split's queue is full, and the ACD split does not have coverage.
- When a call encounters a busy vector command in vector processing.
- When a Direct-Agent call tries to terminate an on-PBX ACD agent and the specified ACD agent's split queue is full and the specified ACD agent does not have coverage.
- When a call is trying to reach an off-PBX party and an ISDN DISConnect message with a User Busy cause is received from an ISDN-PRI facility.

The Failed Event Report is also generated when the destination of a call receives reorder/denial treatment, as follows:

- When a call is trying to terminate to an on-PBX destination but the destination specified is inconsistent with the dial plan, has failed the "class of restriction" check, or inter-digit timeout has occurred.
- When a call encounters a step in vector processing which causes the denial treatment to be applied to the originator.
- When a Direct-Agent call is placed to a destination agent who is not a member of the specified split.
- When a Direct-Agent call is placed to a destination agent who is not logged in.

The Failed Event Report is not sent under the following circumstances:

- For a `cstaMakePredictiveCall` call when any of the above conditions occur the Call Cleared Event Report is generated to indicate that the call has been terminated.

The call is terminated because a connection could not be established to the destination.

Service Parameters:

<i>monitorCrossRefID</i>	[mandatory] Contains the handle to the monitor request for which this event is reported.
<i>failingConnection</i>	[mandatory — partially supported] Specifies the callID of the call that failed
<i>failingDevice</i>	[mandatory — partially supported] Specifies the device that failed. The deviceIDStatus may be ID_NOT_KNOWN.
<i>calledDevice</i>	[mandatory — partially supported] Specifies the called device. The following rules apply: <ul style="list-style-type: none">■ For outgoing calls over PRI facilities, the “called number” from the ISDN SETUP message is specified. If the “called number” does not exist (it is NULL), the deviceIDStatus is ID_NOT_KNOWN.■ For outgoing calls over non-PRI facilities, then the deviceIDStatus is ID_NOT_KNOWN.■ For calls to a TEG (principal) group, the TEG group extension is provided.■ If the busy party is on the PBX, then the extension of the party will be specified. If there is an internal error in the extension, then the deviceIDStatus is ID_NOT_KNOWN.■ For incoming calls to a principal with bridges, the principal’s extension is provided.■ If the destination is inconsistent with the dial plan, then the deviceIDStatus is ID_NOT_KNOWN.
<i>localConnectionInfo</i>	[optional — supported] Specifies the local connection state as perceived by the monitored device on this call. This is provided for the cstaMonitorDevice requests only. A value of CS_NONE indicates that the local connection state is unknown.
<i>cause</i>	[optional — supported] Specifies the reason for this event. The following Event Causes are explicitly sent from the switch: <ul style="list-style-type: none">■ EC_BUSY — User is busy or queue is full.■ EC_CALL_NOT_ANSWERED — User is not responding.■ EC_TRUNKS_BUSY — No trunks are available.■ EC_RESOURCES_NOT_AVAILABLE — Call cannot be completed due to switching resources limitation; for example, no circuit or channel is available.

- EC_REORDER_TONE — Call is rejected or outgoing call is barred.
- EC_DEST_NOT_OBTAINABLE — Invalid destination number.
- EC_NETWORK_NOT_OBTAINABLE — Bearer capability is not available.
- EC_INCOMPATIBLE_DESTINATION — Incompatible destination number. For example, a call from a voice station to a data extension.
- EC_NO_AVAILABLE_AGENTS — Queue full or for direct agent calls — the agent is not a member of the split or the agent is not logged in.

Detailed Information:

See the “Event Report Detailed Information” section in this chapter.

Syntax

```
#include <acs.h>
#include <csta.h>

// CSTAFailedEvent

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTAUNSOLICITED
    EventType_t eventType; // CSTA_FAILED
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            CSTAMonitorCrossRefID_t monitorCrossRefId;
            union
            {
                CSTAFailedEvent_t failed;
            } u;
        } cstaUnsolicited;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAFailedEvent_t
{
    ConnectionID_t failedConnection;
    SubjectDeviceID_t failingDevice;
    CalledDeviceID_t calledDevice;
    LocalConnectionState_t localConnectionInfo;
    CSTAEventCause_t cause;
} CSTAFailedEvent_t;
```

Held Event

Direction: Switch to Client

Event: CSTAHeldEvent

Service Parameters: monitorCrossRefID, heldConnection, holdingDevice, localConnectionInfo, cause

Functional Description:

The Held Event Report indicates that an on-PBX station has placed a call on hold. This includes the hold for conference and transfer.



Placing a call on hold can be done either manually at the station or via a Hold Service request.

Service Parameters:

<i>monitorCrossRefID</i>	[mandatory] Contains the handle to the monitor request for which this event is reported.
<i>heldConnection</i>	[mandatory] Specifies the endpoint where hold was activated.
<i>holdingDevice</i>	[mandatory] Specifies the station extension that placed the call on hold.
<i>localConnectionInfo</i>	[optional — supported] Specifies the local connection state as perceived by the monitored device on this call. This is provided for cstaMonitorDevice requests only. A value of CS_NONE indicates that the local connection state is unknown.
<i>cause</i>	[optional — supported] Specifies the cause for this event. The following causes are supported. <ul style="list-style-type: none"> ■ EC_KEY_CONFERENCE — Indicates that the event report occurred at a bridged device. ■ EC_NEW_CALL — The call has not yet been transferred.

Detailed Information:

See the “Event Report Detailed Information” section in this chapter.

Syntax

```
#include <acs.h>
#include <csta.h>

// CSTAHeldEvent

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTAUNSOLICITED
    EventType_t eventType; // CSTA_HELD
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            CSTAMonitorCrossRefID_t monitorCrossRefId;
            union
            {
                CSTAHeldEvent_t held;
            } u;
        } cstaUnsolicited;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAHeldEvent_t
{
    ConnectionID_t heldConnection;
    SubjectDeviceID_t holdingDevice;
    LocalConnectionState_t localConnectionInfo;
    CSTAEventCause_t cause;
} CSTAHeldEvent_t;
```

Logged Off Event

Direction: Switch to Client

Event: CSTALoggedOffEvent

Private Data Event: ATTLoggedOffEvent

Service Parameters: monitorCrossRefID, agentDevice, agentID, agent

GroupPrivate Parameters: reasonCode

Functional Description:

The Logged Off Event Report informs the application that an agent has logged out of an ACD Split. An application needs to request a cstaMonitorDevice on the ACD Split in order to receive this event.

Service Parameters:

monitorCrossRefID [mandatory] Contains the handle to the monitor request for which this event is reported.

agentDevice [mandatory] Indicates the extension of the agent that is logging on.

agentID [optional — not supported] Indicates the agent identifier.

agentGroup [optional — supported] Indicates the ACD Split that is being logged on.

Private Parameters:

reasonCode [optional] Specifies the reason the agent logged out. Valid reason codes are a single digit 1– 9. A value of 0 indicates that the reason code is not available. The meaning of the code (1-9) is defined by the DEFINITY switch. This parameter is supported by private data version 5 and later only.

Detailed Information:

See the “Event Report Detailed Information” section in this chapter.

Syntax

```

#include <acs.h>
#include <csta.h>

// CSTALoggedOffEvent

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTAUNSOLICITED
    EventType_t eventType; // CSTA_LOGGED_OFF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            CSTAMonitorCrossRefID_t monitorCrossRefId;
            union
            {
                CSTALoggedOffEvent_t loggedOff;
            } u;
        } cstaUnsolicited;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTALoggedOffEvent_t
{
    SubjectDeviceID_t agentDevice;
    AgentID_t agentID;
    AgentGroup_t agentGroup;
} CSTALoggedOffEvent_t;

typedef ExtendedDeviceID_t SubjectDeviceID_t;

typedef char AgentID_t[32];

typedef DeviceID_t AgentGroup_t;

```

Private Parameter Syntax

If private data accompanies a CSTALoggedOffEvent, then the private data would be stored in the location that the application specified as the privateData parameter in the acsGetEventBlock() or acsGetEventPoll() request. If the privateData pointer is set to NULL in these requests, then the CSTALoggedOffEvent does not deliver private data to the application. If the acsGetEventBlock() or acsGetEventPoll() returns a Private Data length of 0, then no private data is provided with this event.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTLoggedOffEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_teventType;// ATT_LOGGED_OFF
    union
    {
        ATTLoggedOffEvent_tloggedOff;
    } u;
} ATTEvent_t;

typedef struct ATTLoggedOffEvent_t
{
    long    reasonCode;// single digit 1 - 9
} ATTLoggedOffEvent_t;
```

Logged On Event

Direction: Switch to Client
Event: CSTALoggedOnEvent
Private Data Event: ATTLoggedOnEvent
Service Parameters: monitorCrossRefID, agentDevice, agentID, agentGroup, password
Private Parameters: workMode

Functional Description:

The Logged On Event Report informs the application that an agent has logged into an ACD Split. An application needs to request a cstaMonitorDevice on the ACD Split in order to receive this event.

The initial agent work mode is provided in the private data.

Service Parameters:

<i>monitorCrossRefID</i>	[mandatory] Contains the handle to the monitor request for which this event is reported.
<i>agentDevice</i>	[mandatory] Indicates the station extension of the agent that is logging on.
<i>agentID</i>	[optional — partially supported] Indicates the logical agent identifier. This is provided for an EAS environment only. For a traditional ACD environment, this is not supported.
<i>agentGroup</i>	[optional — supported] Indicates the ACD Split that is being logged on.
<i>password</i>	[optional — not supported] Indicates the agent password for logging in.

Private Parameters:

<i>workMode</i>	[optional — not supported] Specifies the initial work mode for the Agent as Auxiliary-Work Mode (WM_AUX_WORK), After-Call-Work Mode (WM_AFT_CALL), Auto-In Mode (WM_AUTO_IN), or Manual-In-Work Mode (WM_MANUAL_IN).
------------------------	--

Detailed Information:

In addition to the information provided below, see the “Event Report Detailed Information” section in this chapter.

- Service Availability — This event is only available on a G3 PBX with G3V4 or later software.

Syntax

```
#include <acs.h>
#include <csta.h>

// CSTALoggedOnEvent

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTAUNSOLICITED
    EventType_t eventType; // CSTA_LOGGED_ON
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            CSTAMonitorCrossRefID_t monitorCrossRefId;
            union
            {
                CSTALoggedOnEvent_t loggedOn;
            } u;
        } cstaUnsolicited;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTALoggedOnEvent_t
{
    SubjectDeviceID_t agentDevice;
    AgentID_t agentID;
    AgentGroup_t agentGroup;
    AgentPassword_t password; // not supported
} CSTALoggedOnEvent_t;

typedef ExtendedDeviceID_t SubjectDeviceID_t;

typedef char AgentID_t[32];

typedef DeviceID_t AgentGroup_t;

typedef char AgentPassword_t[32];
```


Private Parameter Syntax

If private data accompanies a CSTALoggedOnEvent, then the private data would be stored in the location that the application specified as the privateData parameter in the acsGetEventBlock() or acsGetEventPoll() request. If the privateData pointer is set to NULL in these requests, then the CSTALoggedOnEvent does not deliver private data to the application. If the acsGetEventBlock() or acsGetEventPoll() returns a Private Data length of 0, then no private data is provided with this event.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTLoggedOnEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_teventType;// ATT_LOGGED_ON
    union
    {
        ATTLoggedOnEvent_tloggedOnEvent;
    } u;
} ATTEvent_t;

typedef struct ATTLoggedOnEvent_t
{
    ATTWorkMode_tworkMode;
} ATTLoggedOnEvent_t;

typedef enum ATTWorkMode_t {
    WM_AUX_WORK          = 1,
    WM_AFTCAL_WK         = 2,
    WM_AUTO_IN           = 3,
    WM_MANUAL_IN         = 4
} ATTWorkMode_t;
```

Network Reached Event

Direction: Switch to Client

Event: CSTANetworkReachedEvent

Private Data Event: ATTNetworkReachedEvent (private data version 5),
ATTV4NetworkReachedEvent (private data versions 2, 3, and 4)

Service Parameters: monitorCrossRefID, connection, trunkUsed,
calledDevice, localConnectionInfo, cause

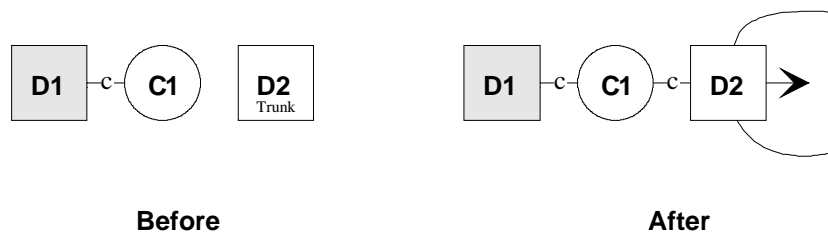
Private Parameters: progressLocation, progressDescription, trunkGroup,
trunkMember

Functional Description:

This event indicates the following two situations when establishing a connection:

- a non-ISDN call is cut through the switch boundary to another network (set to outgoing trunk), or
- an ISDN call is leaving the ISDN network.

Switching Subdomain Boundary



This event report implies that there will be a reduced level of event reporting and possibly no additional device feedback, except disconnect/drop, provided for this party in the call. A Network Reached Event Report is never sent for calls made to devices connected directly to the switch.

The Network Reached Event Report is generated when:

- an ISDN PROG (ISDN progress) message has been received for a call using the ISDN-PRI facilities.²
- a call is placed to an off-PBX destination and a non-PRI trunk is seized
- a call is redirected to an off-PBX destination and a non-PRI trunk is seized.

A switch may receive multiple PROGRESS messages for any given call; each will generate a Network Reached Event Report. This event will not be generated for a cstaMakePredictiveCall call.

2. The reason for the PROG (progress) message is contained in the Progress Indicator. This indicator is sent in private data.

Service Parameters:

<i>monitorCrossRefID</i>	[mandatory] Contains the handle to the monitor request for which this event is reported.
<i>connection</i>	[mandatory] Specifies the endpoint for the outbound connection to another network.
<i>trunkUsed</i>	[mandatory — not supported] Specifies the trunk identifier that was used to establish the connection. This information is provided in the private data.
<i>calledDevice</i>	[mandatory — partially supported] Specifies the destination device of the call. The deviceIDStatus may be ID_NOT_KNOWN.
<i>localConnectionInfo</i>	[optional — supported] Specifies the local connection state as perceived by the monitored device on this call. This is provided for cstaMonitorDevice requests only. A value of CS_NONE indicates that the local connection state is unknown.
<i>cause</i>	[optional — supported] Specifies the cause for this event. The following cause is supported. <ul style="list-style-type: none">■ EC_REDIRECTED — The call has been redirected.

Private Parameters:

<i>progressLocation</i>	[mandatory] Specifies the progress location in a Progress Indicator Information Element from the PRI network. The following location indicators are supported: <ul style="list-style-type: none">■ PL_USER■ PL_PUB_LOCAL■ PL_PUB_REMOTE■ PL_PRIV_REMOTE
<i>progressDescription</i>	[mandatory] Specifies the progress description in a Progress Indicator Information Element from the PRI network. The following description indicators are supported: <ul style="list-style-type: none">■ PD_CALL_OFF_ISDN■ PD_DEST_NOT_ISDN■ PD_ORIG_NOT_ISDN■ PD_CALL_ON_ISDN

- PD_INBAND

trunkGroup [optional — limited supported] This parameter is supported by G3V6 and later switches only. Specifies the trunk group number from which the call leaves the switch and enters the network. This information will not be reported in the originalCallInfo parameter in the events following Network Reached. This parameter is supported by private data version 5 and later only.

trunkMember [optional — limited supported] This parameter is supported by G3V6 and later switches only. Specifies the trunk member from which the call leaves the switch and enters the network. This information will not be reported in the originalCallInfo parameter in the events following Network Reached. This parameter is supported by private data version 5 and later only.

Detailed Information:

See the “Event Report Detailed Information” section in this chapter.

Syntax

```
#include <acs.h>
#include <csta.h>

// CSTANetworkReachedEvent

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTAUNSOLICITED
    EventType_t eventType; // CSTA_NETWORK_REACHED
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            CSTAMonitorCrossRefID_t monitorCrossRefId;
            union
            {
                CSTANetworkReachedEvent_t networkReached;
            } u;
        } cstaUnsolicited;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTANetworkReachedEvent_t
{
    ConnectionID_t connection;
    SubjectDeviceID_t trunkUsed;
    CalledDeviceID_t calledDevice;
    LocalConnectionState_t localConnectionInfo;
    CSTAEventCause_t cause;
} CSTANetworkReachedEvent_t;
```

Private Data Version 5 Syntax

If private data accompanies a `CSTANetworkReachedEvent`, then the private data would be stored in the location that the application specified as the `privateData` parameter in the `acsGetEventBlock()` or `acsGetEventPoll()` request. If the `privateData` pointer is set to `NULL` in these requests, then the `CSTANetworkReachedEvent` does not deliver private data to the application. If `acsGetEventBlock()` or `acsGetEventPoll()` returns a Private Data length of 0, then no private data is provided with this event.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTNetworkReachedEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_t eventType; // ATT_NETWORK_REACHED
    union
    {
        ATTNetworkReachedEvent_t networkReachedEvent;
    } u;
} ATTEvent_t;

typedef struct ATTNetworkReachedEvent_t
{
    ATTProgressLocation_t progressLocation;
    ATTProgressDescription_t progressDescription;
    DeviceID_t trunkGroup;
    DeviceID_t trunkMember;
} ATTNetworkReachedEvent_t;

// ATT progress location values

typedef enum ATTProgressLocation_t
{
    PL_USER = 0, // user
    PL_PUB_LOCAL = 1, // public network serving
                    // local user
    PL_PUB_REMOTE = 4, // public network serving
                    // remote user
    PL_PRIV_REMOTE = 5 // private network serving
                    // remote user
} ATTProgressLocation_t;
```

Private Data Version 5 Syntax (Continued)

```
// ATT progress description values

typedef enum ATTProgressDescription_t
{
    PD_CALL_OFF_ISDN= 1, // call is not end-to-end ISDN,
                        // call progress in-band
    PD_DEST_NOT_ISDN= 2, // destination address is
                        // non-ISDN
    PD_ORIG_NOT_ISDN= 3, // origination address is non-ISDN
    PD_CALL_ON_ISDN = 4, // call has returned to ISDN
    PD_INBAND       = 8 // in-band information now
                        // available
} ATTProgressDescription_t;
```

Private Data Versions 2-4 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV4NetworkReachedEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_t eventType; // ATTV4_NETWORK_REACHED
    union
    {
        ATTV4NetworkReachedEvent_t tv4networkReachedEvent;
    } u;
} ATTEvent_t;

typedef struct ATTV4NetworkReachedEvent_t
{
    ATTProgressLocation_t    progressLocation;
    ATTProgressDescription_t progressDescription;
} ATTV4NetworkReachedEvent_t;

// ATT progress location values

typedef enum ATTProgressLocation_t
{
    PL_USER          = 0,    // user
    PL_PUB_LOCAL     = 1,    // public network serving
                        // local user
    PL_PUB_REMOTE    = 4,    // public network serving
                        // remote user
    PL_PRIV_REMOTE   = 5     // private network serving
                        // remote user
} ATTProgressLocation_t;

// ATT progress description values

typedef enum ATTProgressDescription_t
{
    PD_CALL_OFF_ISDN= 1,    // call is not end-to-end ISDN,
                        // call progress in-band
    PD_DEST_NOT_ISDN= 2,    // destination address is
                        // non-ISDN
    PD_ORIG_NOT_ISDN= 3,    // origination address is non-ISDN
    PD_CALL_ON_ISDN  = 4,    // call has returned to ISDN
    PD_INBAND        = 8     // in-band information now
                        // available
} ATTProgressDescription_t;
```


Originated Event

Direction: Switch to Client
Event: CSTAOriginatedEvent
Private Data Event: ATTOrientedEvent (private data version 6),
ATTV5OriginatedEvent (private data version 2, 3, 4, and 5)
Service Parameters: monitorCrossRefID, originatedConnection,
callingDevice, calledDevice, localConnectionInfo, cause
Private Parameters: logicalAgent, userInfo

Functional Description:

The Originated Event Report indicates that a station has completed dialing and the switch has decided to attempt the call. This event is reported to cstaMonitorDevice associations only.



This event is generated as follows:

- When a station user completes dialing a valid number.
- When a cstaMakeCall is requested on a station, and the station is in the off-hook state (goes off-hook manually, or is forced off-hook), the switch processes the request and determines that a call is to be attempted.
- When a call is attempted using an outgoing trunk and the switch stops collecting digits for that call.

This event will not be reported when a call is aborted because an invalid number was provided, or because the originating number provided is not allowed (via COR) to originate a call.

Service Parameters:

<i>monitorCrossRefID</i>	[mandatory] Contains the handle to the monitor request for which this event is reported.
<i>originatedConnection</i>	[mandatory] Specifies the connection for which the call has been originated.
<i>callingDevice</i>	[mandatory] Specifies the device from which the call has been originated.
<i>calledDevice</i>	[mandatory] Specifies the number that the user dialed or the destination requested by a <code>cstaMakeCall</code> . This is the number dialed rather than the number out-pulsed. It does not include the AAR/ARS FAC (Feature Access Code), or TAC (Trunk Access Code; for example, without the leading 9 often used as the ARS FAC).
<i>localConnectionInfo</i>	[optional — supported] Specifies the local connection state as perceived by the monitored device on this call. This information is provided for <code>cstaMonitorDevice</code> requests only. A value of <code>CS_NONE</code> indicates that the local connection state is unknown.
<i>cause</i>	[optional — supported] Specifies the cause for this event. The following causes are supported: <ul style="list-style-type: none">■ <code>EC_KEY_CONFERENCE</code> — Indicates that the event report occurred at a bridged device. This cause has higher precedence than the following cause.■ <code>EC_NEW_CALL</code> — The call has not yet been redirected.

Private Parameters:

<i>logicalAgent</i>	[optional] Specifies the logical agent extension of the agent that is logged into the station making the call for a <code>cstaMakeCall</code> request.
<i>userInfo</i>	[optional] This parameter allows the application to associate caller information, up to 32 or 96 bytes, with a call. This information may be a customer number, credit card number, alphanumeric digits, or a binary string.

⇒ NOTE:

The `userInfo` parameter is defined for this event, but it is not supported by the DEFINITY switch (i.e., the `userInfo` parameter will not be received for this event).

Detailed Information:

In addition to the information provided below, see the “Event Report Detailed Information” section in this chapter.

- **Abbreviated Dialing** — The Originated Event will be reported when a call is attempted after requesting an abbreviated or speed dialing feature.
- **Account Codes** — (CDR or SMDR Account Code Dialing) — The Originated Event will be reported when a call is originated after an optional or mandatory account code entry.
- **Authorization Codes** — The Originated Event will be reported when a call is originated after an authorization code entry.
- **Automatic Callback** — The Originated Event will be reported when an automatic callback feature matures and the caller goes off-hook on the automatic callback call.
- **Bridged Call Appearance** — The Originated Event will be reported for a call originated from a bridged appearance.
- **Call Park** — The Originated Event will not be reported when a call is parked or retrieved from a parking spot.
- **cstaMakePredictiveCall** — The Originated Event will not be reported for a cstaMakePredictiveCall.
- **Service Availability** — This event is only available on a G3 PBX with G3V4 or later software.

Syntax

```
#include <acs.h>
#include <csta.h>

// CSTAOriginatedEvent

typedef struct
{
    ACSHandle_t      acsHandle;
    EventClass_t    eventClass; // CSTAUNSOLICITED
    EventType_t     eventType;  // CSTA_ORIGINATED
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            CSTAMonitorCrossRefID_t monitorCrossRefId;
            union
            {
                CSTAOriginatedEvent_t originated;
            } u;
        } cstaUnsolicited;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAOriginatedEvent_t
{
    ConnectionID_t      originatedConnection;
    SubjectDeviceID_t   callingDevice;
    CalledDeviceID_t    calledDevice;
    LocalConnectionState_t localConnectionInfo;
    CSTAEventCause_t    cause;
} CSTAOriginatedEvent_t;
```

Private Data Version 6 Syntax

If private data accompanies a CSTAOriginatedEvent, then the private data would be stored in the location that the application specified as the privateData parameter in the acsGetEventBlock() or acsGetEventPoll() request. If the privateData pointer is set to NULL in these requests, then the CSTAOriginatedEvent does not deliver private data to the application. If the acsGetEventBlock() or acsGetEventPoll() returns a Private Data length of 0, then no private data is provided with this event.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTOrganizedEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_t    eventType;    // ATT_ORIGINALATED
    union
    {
        ATTOrganizedEvent_t    originatedEvent;
    } u;
} ATTEvent_t;

typedef struct ATTOrganizedEvent_t
{
    DeviceID_t        logicalAgent;
    ATTUserToUserInfo_t    userInfo;
} ATTOrganizedEvent_t;

typedef struct ATTUserToUserInfo_t {
    ATTUIProtocolType_t    type;
    struct {
        short                length;    // 0 indicates UII not
                                    // present
        unsigned char        value[33];
    } data;
} ATTUserToUserInfo_t;

typedef enum ATTUIProtocolType_t {
    UII_NONE            = -1, // indicates not specified
    UII_USER_SPECIFIC  = 0, // user-specific
    UII_IA5_ASCII       = 4 // null terminated ascii
                        // character string
} ATTUIProtocolType_t;
```

Private Data Version 2-5 Syntax

If private data accompanies a CSTAOriginatedEvent, then the private data would be stored in the location that the application specified as the privateData parameter in the acsGetEventBlock() or acsGetEventPoll() request. If the privateData pointer is set to NULL in these requests, then the CSTAOriginatedEvent does not deliver private data to the application. If the acsGetEventBlock() or acsGetEventPoll() returns a Private Data length of 0, then no private data is provided with this event.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTOrganizedEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_t    eventType;    // ATT_ORIGINATED
    union
    {
        ATTOrganizedEvent_t    originatedEvent;
    } u;
} ATTEvent_t;

typedef struct ATTOrganizedEvent_t
{
    DeviceID_t        logicalAgent;
    ATTUserToUserInfo_t    userInfo;
} ATTOrganizedEvent_t;

typedef struct ATTV5UserToUserInfo_t {
    ATTUUIProtocolType_t    type;
    struct {
        short                length;    // 0 indicates UII not
                                    // present
        unsigned char        value[33];
    } data;
} ATTV5UserToUserInfo_t;

typedef enum ATTUUIProtocolType_t {
    UII_NONE                = -1, // indicates not specified
    UII_USER_SPECIFIC       = 0, // user-specific
    UII_IA5_ASCII           = 4  // null terminated ascii
                                // character string
} ATTUUIProtocolType_t;
```

Queued Event

Direction: Switch to Client

Event: CSTAQueuedEvent

Service Parameters: monitorCrossRefID, queuedConnection, queue, callingDevice, calledDevice, lastRedirectionDevice, numberQueued, localConnectionInfo, cause

Functional Description:

The Queued Event Report indicates that a call queued.



The Queued Event report is generated as follows:

- When a cstaMakePredictiveCall call is delivered to a hunt group or ACD split and the call queues.
- When a call is delivered or redirected to a hunt group or ACD split and the call queues.

It is possible to have multiple Queued Event Reports for a call. For example, the call vectoring feature may queue a call in up to three ACD splits at any one time. In addition, the event is sent if the call queues to the same split with a different priority.

This event report is not generated when a call queues to an announcement, Vector announcement or trunk group. It is also not generated when a call queues, again, to the same ACD split at the same priority.

Service Parameters:

<i>monitorCrossRefID</i>	[mandatory] Contains the handle to the monitor request for which this event is reported.
<i>queuedConnection</i>	[mandatory] Specifies the connection that queued.
<i>queue</i>	[mandatory] Specifies the queuing device to which the call has queued. This is the extension of the ACD split to which the call queued.
<i>callingDevice</i>	[mandatory — partially supported] Specifies the calling device. The deviceIDStatus may be ID_NOT_KNOWN.
<i>calledDevice</i>	[mandatory — partially supported] Specifies the called device. The following rules apply: <ul style="list-style-type: none">■ For incoming calls over PRI facilities, the “called number” from the ISDN SETUP message is specified. If the “called number” does not exist (i.e., NULL), the deviceIDStatus is ID_NOT_KNOWN.■ For incoming calls over non-PRI facilities the called number is the principal extension (a group extension for TEG, PCOL, hunt group, VDN). If the switch is administered to modify the DNIS digits, then the modified DNIS is specified.■ For outbound calls, dialed number is specified.
<i>lastRedirectionDevice</i>	[optional — limited support] Specifies the previous redirection/alerted device in case where the call was redirected/diverted to the queue device.
<i>localConnectionInfo</i>	[optional — supported] Specifies the local connection state as perceived by the monitored device on this call. This is provided for cstaMonitorDevice requests only. A value of CS_NONE indicates that the local connection state is unknown.
<i>numberQueued</i>	[optional — supported] Specifies how many calls are queued to the queue device. This is the call position in the queue in the hunt group or ACD split. This number will include the current call and excludes all direct-agent calls in the queue.
<i>cause</i>	[optional — supported] Specifies the cause for this event. The following cause is supported: <ul style="list-style-type: none">■ EC_REDIRECTED — The call has been redirected.

Detailed Information:

In addition to the information provided below, see the “Event Report Detailed Information” section in this chapter.

- Last Redirection Device — There is only limited support for this parameter. An application must understand the limitations of this parameter in order to use the information correctly.

Syntax

```
#include <acs.h>
#include <csta.h>

// CSTAQueuedEvent

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTAUNSOLICITED
    EventType_t eventType; // CSTA_QUEUED
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            CSTAMonitorCrossRefID_t monitorCrossRefId;
            union
            {
                CSTAQueuedEvent_t queued;
            } u;
        } cstaUnsolicited;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAQueuedEvent_t
{
    ConnectionID_t        queuedConnection;
    SubjectDeviceID_t    queue;
    CallingDeviceID_t    callingDevice;
    CalledDeviceID_t     calledDevice;
    RedirectionDeviceID_t lastRedirectionDevice;
    short                numberQueued;
    LocalConnectionState_t localConnectionInfo;
    CSTAEventCause_t     cause;
} CSTAQueuedEvent_t;
```

Retrieved Event

Direction: Switch to Client

Event: CSTARetrievedEvent

Service Parameters: monitorCrossRefID, retrievedConnection, retrievingDevice, localConnectionInfo, cause

Functional Description:

The Retrieved Event Report indicates that the switch detects a previously held call that has been retrieved.



It is generated when an on-PBX station connects to a call that has been previously placed on hold. Retrieving to a held call can be done either manually at the station by selecting the call appearance of the held call or by switch-hook flash from an analog station, or via a cstaRetrieveCall Service request from a client application.

Service Parameters:

- monitorCrossRefID*** [mandatory] Contains the handle to the monitor request for which this event is reported.
- retrievedConnection*** [mandatory] Specifies the connection for which the call has been taken off the hold state.
- retrievingDevice*** [mandatory] Specifies the device that connected the call from the hold state. This is the extension that has been connected the call.
- localConnectionInfo*** [optional — supported] Specifies the local connection state as perceived by the monitored device on this call. This is provided for *cstaMonitorDevice* requests only. A value of *CS_NONE* indicates that the local connection state is unknown.
- cause*** [optional — supported] Specifies the cause for this event. The following cause is supported:
- *EC_KEY_CONFERENCE* — Indicates that the event report occurred at a bridged device.

Detailed Information:

See the “Event Report Detailed Information” section in this chapter.

Syntax

```
#include <acs.h>
#include <csta.h>

// CSTARetrievedEvent

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTAUNSOLICITED
    EventType_t eventType; // CSTA_RETRIEVED
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            CSTAMonitorCrossRefID_t monitorCrossRefId;
            union
            {
                CSTARetrievedEvent_t retrieved;
            } u;
        } cstaUnsolicited;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTARetrievedEvent_t
{
    ConnectionID_t      retrievedConnection;
    SubjectDeviceID_t  retrievingDevice;
    LocalConnectionState_t localConnectionInfo;
    CSTAEventCause_t   cause;
} CSTARetrievedEvent_t;
```

Service Initiated Event

Direction: Switch to Client

Event: CSTAServiceInitiatedEvent

Private Data Event: ATTServiceInitiatedEvent

Service Parameters: monitorCrossRefID, initiatedConnection,
localConnectionInfo, cause

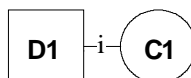
Private Parameters: ucid

Functional Description:

The Service Initiated Event Report indicates that telecommunication service is initiated.



Before



After

This event is generated as follows:

- When a station begins to receive dial tone.
- When a station is forced off-hook because a cstaMakeCall is requested on that station.
- When certain switch features that initiate a call (such as the abbreviated dialing, etc.) are invoked.

Service Parameters:

<i>monitorCrossRefID</i>	[mandatory] Contains the handle to the monitor request for which this event is reported.
<i>initiatedConnection</i>	[mandatory] Specifies the connection for which the service (dial tone) has been initiated.
<i>localConnectionInfo</i>	[optional — supported] Specifies the local connection state as perceived by the monitored device on this call. This is provided for the <code>cstaMonitorDevice</code> requests only. A value of <code>CS_NONE</code> indicates that the local connection state is unknown.
<i>cause</i>	[optional — supported] Specifies the cause for this event. The following cause is supported: <ul style="list-style-type: none">■ <code>EC_KEY_CONFERENCE</code> — Indicates that the event report occurred at a bridged device.

Private Parameters:

<i>ucid</i>	[optional] Specifies the Universal Call ID (UCID) of the resulting call. The UCID is a unique call identifier across switches and the network. A valid UCID is a null-terminated ASCII character string. If there is no UCID associated with this call, the <code>ucid</code> contains the <code>ATT_NULL_UCID</code> (a 20-character string of all zeros). This parameter is supported by private data version 5 and later only.
--------------------	---

Detailed Information:

See the “Event Report Detailed Information” section in this chapter.

Syntax

```
#include <acs.h>
#include <csta.h>

// CSTAServiceInitiatedEvent

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTAUNSOLICITED
    EventType_t eventType; // CSTA_SERVICE_INITIATED
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            CSTAMonitorCrossRefID_t monitorCrossRefId;
            union
            {
                CSTAServiceInitiatedEvent_t serviceInitiated;
            } u;
        } cstaUnsolicited;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTAServiceInitiatedEvent_t
{
    ConnectionID_t          initiatedConnection;
    LocalConnectionState_t localConnectionInfo;
    CSTAEventCause_t       cause;
} CSTAServiceInitiatedEvent_t;
```


Private Parameter Syntax

If private data accompanies a `CSTAServiceInitiatedEvent`, then the private data would be stored in the location that the application specified as the `privateData` parameter in the `acsGetEventBlock()` or `acsGetEventPoll()` request. If the `privateData` pointer is set to `NULL` in these requests, then the `CSTAServiceInitiatedEvent` does not deliver private data to the application. If the `acsGetEventBlock()` or `acsGetEventPoll()` returns a Private Data length of 0, then no private data is provided with this event.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTSERVICEINITIADEVENT - CSTA Unsolicited Event Private Data
// (supported by private data version 5 and later only)

typedef struct
{
    ATTEventTypeeventType; // ATT_SERVICE_INITIATED
    union
    {
        ATTSERVICEINITIADEVENT_tserviceInitiated;
    } u;
} ATTEVENT_t;

typedef struct ATTSERVICEINITIADEVENT_t
{
    ATTUCID_t    ucid;
} ATTSERVICEINITIADEVENT_t;

typedef char ATTUCID_t[64];
```

Transferred Event

Direction: Switch to Client

Event: CSTATransferredEvent

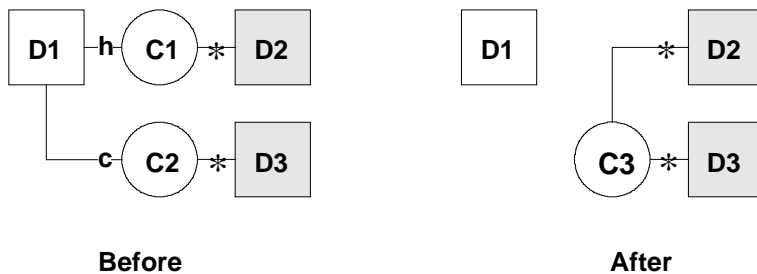
Private Data Event: ATTransferredEvent (private data version 6), ATTV5TransferredEvent (private data version 5), ATTV4TransferredEvent (private data version 4), ATTV3TransferredEvent (private data versions 2 and 3)

Service Parameters: monitorCrossRefID, primaryOldCall, secondaryOldCall, transferringDevice, transferredDevice, transferredConnections, localConnectionInfo, cause

Private Parameters: originalCallInfo, distributingDevice, ucid, trunkList (private data version 6)

Functional Description:

The Transferred Event Report indicates that an existing call was transferred to another device and the device requesting the transfer has been dropped from the call. The transferringDevice will not appear in any future feedback for the call.



The Transferred Event Report is generated for the following circumstances:

- When an on-PBX station completes a transfer by pressing the “transfer” button on the voice terminal.
- When the on-PBX analog set (phone) user on a monitored call goes on hook with one active call and one call on conference/transfer hold.
- When the “call park” feature is used in conjunction with the “transfer” button on the voice set.
- When an adjunct successfully completes a cstaTransferCall request.

Service Parameters:

monitorCrossRefID [mandatory] Contains the handle to the monitor request for which this event is reported.

primaryOldCall [mandatory] Specifies the callID of the call that was transferred. This is usually the held call before the transfer. This call ended as a result of the transfer.

secondaryOldCall [mandatory] Specifies the callID of the call that was transferred. This is usually the active call before the transfer. This call is retained by the switch after the transfer.

transferringDevice [mandatory] Specifies the device that is controlling the transfer. This is the device that did the transfer.

transferredDevice [mandatory] Specifies the new transferred-to device.

- If the device is an on-PBX station, the extension is specified.
- If the party is an off-PBX endpoint, then the deviceIDStatus is ID_NOT_KNOWN.

There are call scenarios in which the transfer operation joins multiple parties to a call. In such situations, the transferredDevice will be the extension for the last party to join the call.

transferredConnections [optional - supported] Specifies a count of the number of devices and a list of connectionIDs and deviceIDs which resulted from the transfer.

- If a device is on-PBX, the extension is specified. The extension consists of station or group extensions. Group extensions are provided when the transfer is to a group and the transfer completes before the call is answered by one of the group members (TEG, PCOL, hunt group, or VDN extension). It may contain alerting extensions.
- The static deviceID of a queued endpoint is set to the split extension of the queue.
- If a party is off-PBX, then its static device identifier or its previously assigned trunk identifier is specified.

localConnectionInfo [optional — supported] Specifies the local connection state as perceived by the monitored device on this call. This is provided for the cstaMonitorDevice requests only. A value of CS_NONE indicates that the local connection state is unknown.

[optional — supported] Specifies the cause for this event.
The following causes are supported:

- EC_TRANSFER — A call transfer has occurred.
- EC_PARK — A call transfer was performed for parking a call rather than a true call transfer operation.

Private Parameters:

originalCallInfo [optional] Specifies the original call information. This parameter is sent with this event for the resulting newCall of a cstaTransferCall request or the retained call of a (manual) transfer call operation. The calls being transferred must be known to the G3PD via the Call Control Services or Monitor Services.

⇒ NOTE:

For a cstaTransferCall, the originalCallInfo includes the call information originally received by the heldCall specified in the cstaTransferCall request. For a manual call transfer, the originalCallInfo includes the call information originally received by the primaryOldCall specified in the event report.

- **reason** — the reason for the originalCallInfo. The following reasons are supported.
 - OR_NONE — no originalCallInfo provided
 - OR_CONFERENCED — call conferenced
 - OR_CONSULTATION — consultation call
 - OR_TRANSFERRED — call transferred
 - OR_NEW_CALL — new call
- **callingDevice** — The original callingDevice received by the heldCall or the primaryOldCall. This parameter is always provided.
- **calledDevice** — The original calledDevice received by the heldCall or the primaryOldCall. This parameter is always provided.
- **trunk** — The original trunk group received by the heldCall or the primaryOldCall. This parameter is supported by private data versions 2, 3, and 4.
- **trunkGroup** — The original trunk group received by the heldCall or the primaryOldCall. This parameter is supported by private data version 5 and later only.
- **trunkMember** (G3V4 switches and later) — The original trunkMember received by the heldCall or the primaryOldCall.
- **lookaheadInfo** — The original lookaheadInfo received by the heldCall or the primaryOldCall.
- **userEnteredCode** — The original userEnteredCode received by the heldCall or the primaryOldCall call.

- **userInfo** — the original userInfo received by the heldCall or the primaryOldCall call.

Prior to G3V8, the maximum length of userInfo was 32 bytes. Beginning with G3V8, the maximum length of userInfo is increased to 96 bytes.

⇒ NOTE:

An application using private data version 5 and earlier can only receive a maximum of 32-byte data for userInfo, regardless of the size data that is sent by the switch.

- **ucid** — the original ucid of the call. This parameter is supported by private data version 5 and later only.
- **callOriginatorInfo** — the original callOriginatorInfo received by the call. This parameter is supported by private data version 5 and later only.
- **flexibleBilling** — the original flexibleBilling information of the call. This parameter is supported by private data version 5 and later only.

See the “Delivered Event” section in this chapter for details on these parameters.

distributingDevice	[optional] specifies the original distributing device before the call is transferred. See the “Delivered Event” section in this chapter for details on the distributingDevice parameter. This parameter is supported by private data version 4 and later.
ucid	[optional] Specifies the Universal Call ID (UCID) of the call. The UCID is a unique call identifier across switches and the network. A valid UCID is a null-terminated ASCII character string. If there is no UCID associated with this call, the ucid contains the ATT_NULL_UCID (a 20-character string of all zeros). This parameter is supported by private data version 5 and later only.
trunkList	[optional] Specifies a list of up to 5 trunk groups and trunk members. This parameter is supported by private data version 6 and later only. The following options are supported: <ul style="list-style-type: none"> ■ count — The count of the connected parties on the call. ■ trunks — An array of 5 trunk group and trunk member IDs, one for each connected party. The following options are supported:

- connection — The connection ID of one of the parties on the call.
- trunkGroup — The trunk group of the party referenced by connection.
- trunkMember — The trunk member of the party referenced by connection.

Detailed Information:

In addition to the information provided below, see the “Event Report Detailed Information” section in this chapter.

The originalCallInfo includes the original call information originally received by the call that is ended as the result of the transfer. The following special rules apply:

- If the Transferred Event was a result of a cstaTransferCall request, the originalCallInfo and the distributingDevice sent with this Transferred Event is from the heldCall in the cstaTransferCall request. Thus the application can control the originalCallInfo and the distributingDevice to be sent in a Transferred Event by putting the original call on hold and specifying it as the heldCall in the cstaTransferCall request. Although the primaryOldCall that is the call ended as the result of the cstaTransferCall is the heldCall most of the time, sometimes it can be the activeCall.
- If the Transferred Event was a result of a manual transfer, the originalCallInfo and the distributingDevice sent with this Transferred Event is from the primaryOldCall of the event. Thus the application does not have control of the originalCallInfo and distributingDevice to be sent in the Transferred Event. Although the primaryOldCall that is the call ended as the result of the manual transfer operation is the heldCall most of the time, sometimes it can be the active call.

In addition, see the Established Event “Detailed Information:” section for Blind Transfer and Consultation Transfer definitions; “Transferring or Conferencing a Call Together with Screen Pop Information” in Chapter 3 for the recommended design for applications that use caller information to populate a screen); and the ANI Screen Pop Application Requirements in the “Event Report Detailed Information” section in this chapter.

Syntax

```
#include <acs.h>
#include <csta.h>

// CSTATransferredEvent

typedef struct
{
    ACSHandle_t    acsHandle;
    EventClass_t  eventClass; // CSTAUNSOLICITED
    EventType_t   eventType;  // CSTA_TRANSFERRED
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            CSTAMonitorCrossRefID_t monitorCrossRefId;
            union
            {
                CSTATransferredEvent_t transferred;
            } u;
        } cstaUnsolicited;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTATransferredEvent_t
{
    ConnectionID_t    primaryOldCall;
    ConnectionID_t    secondaryOldCall;
    SubjectDeviceID_t transferringDevice;
    SubjectDeviceID_t transferredDevice;
    ConnectionList_t transferredConnections;
    LocalConnectionState_t localConnectionInfo;
    CSTAEventCause_t cause;
} CSTATransferredEvent_t;

typedef ExtendedDeviceID_t SubjectDeviceID_t;

typedef struct Connection_t {
    ConnectionID_t    party;
    SubjectDeviceID_t staticDevice;
} Connection_t;

typedef struct ConnectionList_t {
    int    count;
    Connection_t *connection;
} ConnectionList_t;
```


Private Data Version 6 Syntax

If private data accompanies a CSTATransferredEvent, then the private data would be stored in the location that the application specified as the privateData parameter in the acsGetEventBlock() or acsGetEventPoll() request. If the privateData pointer is set to NULL in these requests, then the CSTATransferredEvent does not deliver private data to the application. If acsGetEventBlock() or acsGetEventPoll() returns a Private Data length of 0, then no private data is provided with this event.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTTransferredEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_teventType;// ATT_TRANSFERRED
    union
    {
        ATTTransferredEvent_ttransferredEvent;
    } u;
} ATTEvent_t;

typedef struct ATTTransferredEvent_t
{
    ATTOriginalCallInfo_t    originalCallInfo;
    CalledDeviceID_t        distributingDevice;
    ATTUCID_t                ucid;
} ATTTransferredEvent_t;

typedef struct ATTOriginalCallInfo_t
{
    ATTReasonForCallInfo_t    reason;
    CallingDeviceID_t        callingDevice;
    CalledDeviceID_t        calledDevice;
    DeviceID_t                trunkGroup;
    DeviceID_t                trunkMember;
    ATTLookaheadInfo_t        lookaheadInfo;
    ATTUserEnteredCode_t        userEnteredCode;
    ATTUserToUserInfo_t        userInfo;
    ATTUCID_t                ucid;
    ATTCallOriginatorInfo_t    callOriginatorInfo;
    Boolean                    flexibleBilling;
} ATTOriginalCallInfo_t;
```

Private Data Version 6 Syntax (Continued)

```
typedef enum ATTReasonForCallInfo_t
{
    OR_NONE          = 0, // indicates info not present
    OR_CONSULTATION = 1,
    OR_CONFERENCED  = 2,
    OR_TRANSFERRED  = 3,
    OR_NEW_CALL     = 4
} ATTReasonForCallInfo_t;

typedef ExtendedDeviceID_t CallingDeviceID_t;

typedef ExtendedDeviceID_t CalledDeviceID_t;

typedef struct ATTLookaheadInfo_t
{
    ATTInterflow_t    type;
    ATTPriority_t    priority;
    short             hours;
    short             minutes;
    short             seconds;
    DeviceID_t       sourceVDN;
    ATTUnicodeDeviceID_t sourceVDN;
                    // sourceVDN in Unicode
} ATTLookaheadInfo_t;

typedef enum ATTInterflow_t
{
    LAI_NO_INTERFLOW= -1, // indicates info not present
    LAI_ALL_INTERFLOW= 0,
    LAI_THRESHOLD_INTERFLOW= 1,
    LAI_VECTORING_INTERFLOW= 2
} ATTInterflow_t;

typedef enum ATTPriority_t
{
    LAI_NOT_IN_QUEUE= 0,
    LAI_LOW          = 1,
    LAI_MEDIUM       = 2,
    LAI_HIGH         = 3,
    LAI_TOP          = 4
} ATTPriority_t;

typedef struct ATTUnicodeDeviceID_t
{
    short            count;
    unsigned short  value[64];
} ATTUnicodeDeviceID_t;
```

Private Data Version 6 Syntax (Continued)

```

typedef struct ATTUserEnteredCode_t
{
    ATTUserEnteredCodeType_t    type;
    ATTUserEnteredCodeIndicator_t indicator;
    char                        data[ATT_MAX_USER_CODE];
    DeviceID_t                  collectVDN;
} ATTUserEnteredCode_t;

typedef enum ATTUserEnteredCodeType_t
{
    UE_NONE          = -1, // indicates not specified
    UE_ANY           = 0,
    UE_LOGIN_DIGITS  = 2,
    UE_CALL_PROMPTER = 5,
    UE_DATA_BASE_PROVIDED = 17,
    UE_TONE_DETECTOR = 32
} ATTUserEnteredCodeType_t;

typedef enum ATTUserEnteredCodeIndicator_t
{
    UE_COLLECT = 0,
    UE_ENTERED = 1
} ATTUserEnteredCodeIndicator_t;

typedef struct ATTUserToUserInfo_t
{
    ATTUIProtocolType_t type;
    struct {
        short          length;
        unsigned char  value[ATT_MAX_USER_INFO];
    } data;
} ATTUserToUserInfo_t;

typedef enum ATTUIProtocolType_t
{
    UII_NONE          = -1, // indicates not specified
    UII_USER_SPECIFIC = 0, // user-specific
    UII_IA5_ASCII     = 4
    // null terminated ascii character string
} ATTUIProtocolType_t;

typedef char ATTUCID_t[64];

typedef struct ATTCallOriginatorInfo_t
{
    Boolean hasInfo; // if FALSE, no callOriginatorType
    short   callOriginatorType;
} ATTCallOriginatorInfo_t;

```

Private Data Version 5 Syntax

If private data accompanies a `CSTATransferredEvent`, then the private data would be stored in the location that the application specified as the `privateData` parameter in the `acsGetEventBlock()` or `acsGetEventPoll()` request. If the `privateData` pointer is set to `NULL` in these requests, then the `CSTATransferredEvent` does not deliver private data to the application. If `acsGetEventBlock()` or `acsGetEventPoll()` returns a Private Data length of 0, then no private data is provided with this event.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV5TransferredEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_t eventType; // ATT_TRANSFERRED
    union
    {
        ATTV5TransferredEvent_t transferredEvent;
    } u;
} ATTEvent_t;

typedef struct ATTV5TransferredEvent_t
{
    ATTV5OriginalCallInfo_t originalCallInfo;
    CalledDeviceID_t         distributingDevice;
    ATTUCID_t                ucid;
} ATTV5TransferredEvent_t;

typedef struct ATTV5OriginalCallInfo_t
{
    ATTReasonForCallInfo_t reason;
    CallingDeviceID_t      callingDevice;
    CalledDeviceID_t       calledDevice;
    DeviceID_t             trunkGroup;
    DeviceID_t             trunkMember;
    ATTLookaheadInfo_t     lookaheadInfo;
    ATTUserEnteredCode_t   userEnteredCode;
    ATTV5UserToUserInfo_t  userInfo;
    ATTUCID_t              ucid;
    ATTCallOriginatorInfo_t callOriginatorInfo;
    Boolean                 flexibleBilling;
} ATTV5OriginalCallInfo_t;
```

Private Data Version 5 Syntax (Continued)

```
typedef enum ATTReasonForCallInfo_t
{
    OR_NONE          = 0, // indicates info not present
    OR_CONSULTATION = 1,
    OR_CONFERENCED  = 2,
    OR_TRANSFERRED  = 3,
    OR_NEW_CALL     = 4
} ATTReasonForCallInfo_t;

typedef ExtendedDeviceID_tCallingDeviceID_t;

typedef ExtendedDeviceID_tCalledDeviceID_t;

typedef struct ATTLookaheadInfo_t
{
    ATTInterflow_t    type;
    ATTPriority_t    priority;
    short             hours;
    short             minutes;
    short             seconds;
    DeviceID_t        sourceVDN;
    ATTUnicodeDeviceID_t sourceVDN;
                    // sourceVDN in Unicode
} ATTLookaheadInfo_t;

typedef enum ATTInterflow_t
{
    LAI_NO_INTERFLOW= -1, // indicates info not present
    LAI_ALL_INTERFLOW= 0,
    LAI_THRESHOLD_INTERFLOW= 1,
    LAI_VECTORING_INTERFLOW= 2
} ATTInterflow_t;

typedef enum ATTPriority_t
{
    LAI_NOT_IN_QUEUE= 0,
    LAI_LOW          = 1,
    LAI_MEDIUM       = 2,
    LAI_HIGH         = 3,
    LAI_TOP          = 4
} ATTPriority_t;

typedef struct ATTUnicodeDeviceID_t
{
    short            count;
    unsigned short  value[64];
} ATTUnicodeDeviceID_t;
```

Private Data Version 5 Syntax (Continued)

```
typedef struct ATTUserEnteredCode_t
{
    ATTUserEnteredCodeType_t    type;
    ATTUserEnteredCodeIndicator_t indicator;
    char                        data[ATT_MAX_USER_CODE];
    DeviceID_t                  collectVDN;
} ATTUserEnteredCode_t;

typedef enum ATTUserEnteredCodeType_t
{
    UE_NONE          = -1, // indicates not specified
    UE_ANY           = 0,
    UE_LOGIN_DIGITS = 2,
    UE_CALL_PROMPTER = 5,
    UE_DATA_BASE_PROVIDED= 17,
    UE_TONE_DETECTOR = 32
} ATTUserEnteredCodeType_t;

typedef enum ATTUserEnteredCodeIndicator_t
{
    UE_COLLECT= 0,
    UE_ENTERED= 1
} ATTUserEnteredCodeIndicator_t;

typedef struct ATTV5UserToUserInfo_t
{
    ATTUIProtocolType_t type;
    struct {
        short          length;
        unsigned char  value[33];
    } data;
} ATTV5UserToUserInfo_t;

typedef enum ATTUIProtocolType_t
{
    UII_NONE          = -1, // indicates not specified
    UII_USER_SPECIFIC = 0, // user-specific
    UII_IA5_ASCII     = 4
    // null terminated ascii character string
} ATTUIProtocolType_t;

typedef char ATTUCID_t[64];

typedef struct ATTCallOriginatorInfo_t
{
    Boolean hasInfo; // if FALSE, no callOriginatorType
    short   callOriginatorType;
} ATTCallOriginatorInfo_t;
```

Private Data Version 4 Syntax

```

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV4TransferredEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_t eventType; // ATTV4_TRANSFERRED
    union
    {
        ATTV4TransferredEvent_t tv4transferredEvent;
    } u;
} ATTEvent_t;

typedef struct ATTV4TransferredEvent_t
{
    ATTV4OriginalCallInfo_t originalCallInfo;
    CalledDeviceID_t          distributingDevice;
} ATTV4TransferredEvent_t;

typedef struct ATTV4OriginalCallInfo_t
{
    ATTReasonForCallInfo_t reason;
    CallingDeviceID_t      callingDevice;
    CalledDeviceID_t       calledDevice;
    DeviceID_t             trunk;
    DeviceID_t             trunkMember;
    ATTV4LookaheadInfo_t  lookaheadInfo;
    ATTUserEnteredCode_t  userEnteredCode;
    ATTV5UserToUserInfo_t userInfo;
} ATTV4OriginalCallInfo_t;

typedef enum ATTReasonForCallInfo_t
{
    OR_NONE          = 0, // indicates info not present
    OR_CONSULTATION = 1,
    OR_CONFERENCED  = 2,
    OR_TRANSFERRED  = 3,
    OR_NEW_CALL     = 4
} ATTReasonForCallInfo_t;

typedef ExtendedDeviceID_t CallingDeviceID_t;

typedef ExtendedDeviceID_t CalledDeviceID_t;

```

Private Data Version 4 Syntax (Continued)

```
typedef struct ATTV4LookaheadInfo_t
{
    ATTInterflow_t    type;
    ATTPriority_t     priority;
    short              hours;
    short              minutes;
    short              seconds;
    DeviceID_t        sourceVDN;
} ATTV4LookaheadInfo_t;

typedef enum ATTInterflow_t
{
    LAI_NO_INTERFLOW= -1, // indicates info not present
    LAI_ALL_INTERFLOW= 0,
    LAI_THRESHOLD_INTERFLOW= 1,
    LAI_VECTORING_INTERFLOW= 2
} ATTInterflow_t;

typedef enum ATTPriority_t
{
    LAI_NOT_IN_QUEUE= 0,
    LAI_LOW           = 1,
    LAI_MEDIUM        = 2,
    LAI_HIGH          = 3,
    LAI_TOP           = 4
} ATTPriority_t;

typedef struct ATTUserEnteredCode_t
{
    ATTUserEnteredCodeType_t    type;
    ATTUserEnteredCodeIndicator_t indicator;
    char                         data[ATT_MAX_USER_CODE];
    DeviceID_t                   collectVDN;
} ATTUserEnteredCode_t;

typedef enum ATTUserEnteredCodeType_t
{
    UE_NONE           = -1, // indicates not specified
    UE_ANY            = 0,
    UE_LOGIN_DIGITS  = 2,
    UE_CALL_PROMPTER = 5,
    UE_DATA_BASE_PROVIDED= 17,
    UE_TONE_DETECTOR = 32
} ATTUserEnteredCodeType_t;

typedef enum ATTUserEnteredCodeIndicator_t
{
    UE_COLLECT= 0,
    UE_ENTERED= 1
} ATTUserEnteredCodeIndicator_t;
```


Private Data Version 4 Syntax (Continued)

```
typedef struct ATTV5UserToUserInfo_t
{
    ATTUUIProtocolType_t type;
    struct {
        short length; // 0 indicates UUI not present
        unsigned charvalue[33];
    } data;
} ATTV5UserToUserInfo_t;

typedef enum ATTUUIProtocolType_t
{
    UUI_NONE = -1, // indicates not specified
    UUI_USER_SPECIFIC= 0, // user-specific
    UUI_IA5_ASCII= 4 // null terminated ascii
                    // character string
} ATTUUIProtocolType_t;
```

Private Data Versions 2 and 3 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV3TransferredEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType_t eventType; // ATTV3_TRANSFERRED
    union
    {
        ATTV3TransferredEvent_t tv3transferredEvent;
    } u;
} ATTEvent_t;

typedef struct ATTV3TransferredEvent_t
{
    ATTV4OriginalCallInfo_t originalCallInfo;
} ATTV3TransferredEvent_t;

typedef struct ATTV4OriginalCallInfo_t
{
    ATTReasonForCallInfo_t reason;
    CallingDeviceID_t callingDevice;
    CalledDeviceID_t calledDevice;
    DeviceID_t trunk;
    DeviceID_t trunkMember;
    ATTV4LookaheadInfo_t lookaheadInfo;
    ATTUserEnteredCode_t userEnteredCode;
    ATTV5UserToUserInfo_t userInfo;
} ATTV4OriginalCallInfo_t;

typedef enum ATTReasonForCallInfo_t
{
    OR_NONE = 0, // indicates info not present
    OR_CONSULTATION = 1,
    OR_CONFERENCED = 2,
    OR_TRANSFERRED = 3,
    OR_NEW_CALL = 4
} ATTReasonForCallInfo_t;

typedef ExtendedDeviceID_t CallingDeviceID_t;

typedef ExtendedDeviceID_t CalledDeviceID_t;
```

Private Data Versions 2 and 3 Syntax (Continued)

```

typedef struct ATTV4LookaheadInfo_t
{
    ATTInterflow_t    type;
    ATTPriority_t     priority;
    short             hours;
    short             minutes;
    short             seconds;
    DeviceID_t        sourceVDN;
} ATTV4LookaheadInfo_t;

typedef enum ATTInterflow_t
{
    LAI_NO_INTERFLOW= -1, // indicates info not present
    LAI_ALL_INTERFLOW= 0,
    LAI_THRESHOLD_INTERFLOW= 1,
    LAI_VECTORING_INTERFLOW= 2
} ATTInterflow_t;

typedef enum ATTPriority_t
{
    LAI_NOT_IN_QUEUE = 0,
    LAI_LOW           = 1,
    LAI_MEDIUM        = 2,
    LAI_HIGH           = 3,
    LAI_TOP            = 4
} ATTPriority_t;

typedef struct ATTUserEnteredCode_t
{
    ATTUserEnteredCodeType_t    type;
    ATTUserEnteredCodeIndicator_t indicator;
    char                        data[ATT_MAX_USER_CODE];
    DeviceID_t                  collectVDN;
} ATTUserEnteredCode_t;

typedef enum ATTUserEnteredCodeType_t
{
    UE_NONE           = -1, // indicates not specified
    UE_ANY            = 0,
    UE_LOGIN_DIGITS  = 2,
    UE_CALL_PROMPTER = 5,
    UE_DATA_BASE_PROVIDED= 17,
    UE_TONE_DETECTOR= 32
} ATTUserEnteredCodeType_t;

typedef enum ATTUserEnteredCodeIndicator_t
{
    UE_COLLECT= 0,
    UE_ENTERED= 1
} ATTUserEnteredCodeIndicator_t;

```

Private Data Versions 2 and 3 Syntax (Continued)

```
typedef struct ATTV5UserToUserInfo_t
{
    ATTUUIProtocolType_t type;
    struct {
        short length; // 0 indicates UUI not present
        unsigned charvalue[33];
    } data;
} ATTV5UserToUserInfo_t;

typedef enum ATTUUIProtocolType_t
{
    UUI_NONE = -1, // indicates not specified
    UUI_USER_SPECIFIC = 0, // user-specific
    UUI_IA5_ASCII = 4 // null terminated ascii
                        // character string
} ATTUUIProtocolType_t;
```

Event Report Detailed Information

Analog Sets

Redirection

Analog sets do not support temporary bridged appearances. When, in normal circumstances, a call at a multifunction set would have been left on a simulated bridge appearance, the call will move away from the analog set. Thus, any monitor requests for the analog set will receive the Diverted Event Report.

Delivered Event Reports are not sent to SAC-activated analog sets receiving calls.

Redirection on No Answer

Calls redirected by this feature generate the following event reports when a call is redirected from a nonanswering station.

- Diverted Event Report is provided over the cstaMonitorDevice monitor requests when the call is redirected from a nonanswering agent. This event is not provided if the call is queued again to the split or delivered to another agent in the split.
- Queued Event Report will be generated if the call queues after being redirected.
- Call Cleared Event Report — If the call cannot re-queue after the call has been redirected from the nonanswering agent, then the call continues to listen to ringback until the caller is dropped. In this case, a Call Cleared Event Report is generated when the caller is dropped and the call disconnected.

Direct Agent Calls always redirect to the agent's coverage path instead of queueing again to the servicing ACD split.

Switch Hook Operation

When an analog set goes on-hook with one or two calls on hold, the user is audibly notified (the phone rings). This notification ring is not reported as a Delivered event. When the user goes off-hook and is reconnected to the alerting call, a Retrieved Event Report is generated.

When a user goes on hook with a soft-held call and an active call, both calls are transferred away from the user's set. It does not matter how the held call was placed on soft hold.

If a monitored analog user flashes the switch hook to put a call on soft hold to start a new call:

- The Held Event Report is sent to all monitor requests.
- A Service Initiated Event Report is returned to all `cstaMonitorDevice` requests when the user receives the dial tone.
- A Retrieved Event Report is returned to all monitor requests if the user returns to the held call. If the held call is conferenced or transferred, the Conferenced or Transferred Event Reports are sent to all monitor requests.

ANI Screen Pop Application Requirements

The list below summarizes the prerequisites for ANI screen pop at a station. Each item is discussed in more detail below:

- Incoming PRI provides ANI for incoming external calls. No other external sources (such as “caller-id”) are supported. This is a typical G3 call center configuration.
- Local G3 PBX or DCS provides extension number as ANI for local or private network incoming calls. This is a typical help desk configuration.
- Chapter 3 gives design guidelines for transferring a call across G3PDs, across CTI platforms, and across switches. If these guidelines are not followed, then the transferring party and receiving party must be on the same switch and monitored by the same G3PD. Transfers across a private DCS network are not supported.
- The receiving party may either manually answer the call or run an application that uses `cstaAnswerCall`.

If the design considerations in Chapter 3 are not followed, then ANI screen pop on blind transfer can only be done at the time the call is answered, not when it rings. In this case, applications will find the ANI information in the CSTA Established Event (which the driver sends when the call answers), not the CSTA Delivered Event (which the driver sends when the call rings). For an application to do an ANI screen pop on a blind transfer, it must look in the proper CSTA Event.

If the design considerations in Chapter 3 are not followed, then ANI screen pop on consultation transfer is possible only at the time the call transfers, not when the consultation call rings or is answered. In this case, applications will find the information necessary to do the screen pop in the CSTA Transfer Event (which the driver sends them when the call transfers), not in the CSTA Established or Delivered events. For an application to do an ANI screen pop on a consultation transfer, it must look in the proper CSTA Event.

If the design considerations in Chapter 3 are not followed, then ANI screen pop on a consultation transfer requires that the transferring party must be monitored by the same G3PD that is monitoring the receiving party.

Announcements

Automatic Call Distribution (ACD) split-forced announcements and vector announcements do not generate event reports for the application. However, nonsplit announcements generate events that are sent to other parties on the call.

Extensions assigned to integrated announcements may not be monitored.

Answer Supervision

The G3 PBX “answer supervision timeout” field determines how long the central office trunk board waits before sending the (simulated) “answer” message to the software. This is useful when the answer supervision is not available on a trunk. This message is used to send call information to Station Message Detail Recording (SMDR) and to trigger the bridging of a service observer onto an outgoing trunk call. This message is ignored if the trunk is expected to receive true answer supervision from the network (the switch uses the true answer supervision whenever available). Client application monitored calls are treated like regular calls. No Established Event Report will be generated for this “simulated answer.”

With respect to `cstaMakePredictiveCall` calls, when the “answer supervision” field is set to “no”, the switch relies entirely on the call classifier to determine when the call was answered. When answer supervision on the trunk is set to “yes”, a `cstaMakePredictiveCall` call is considered “answered” when the switch software receives the “answer” message from the trunk board. In reality, `cstaMakePredictiveCall` calls may receive either an “answer” message from the trunk board or (if this never comes) an indication from the classifier that the far end answered. In this case, the switch will act on the first indication received and not act on any subsequent indications.

Attendants and Attendant Groups

An attendant group extension cannot be monitored as a station.

Individual attendants may be parties on monitored calls and are supported like regular station users as far as the event reporting is concerned on monitors for other station types.

An attendant group may be a party on a monitored call, but the Delivered, Established, and Connection Cleared Event Reports do not apply.

An individual attendant extension member cannot be monitored by a `cstaMonitorDevice` request; but it can be a destination for a call from a `cstaMonitorDevice` monitored station. In this case, event reports are sent to the `cstaMonitorDevice` request about the individual attendant that is receiving the call.

Attendant Specific Button Operation

This section clarifies what events are sent when an attendant uses buttons that are specific to an attendant console.

- Hold button — If an individual attendant presses the hold button and the call is monitored, the Held Event Report will be sent to the corresponding monitor request.
- Call Appearance button — If an individual attendant has a call on hold, and the call is monitored, then the Retrieved Event Report will be sent to the corresponding monitor requests.
- Start button — If a call is present at an attendant and the call is monitored, and the attendant presses the start button, then the call will be put on hold and a Held Event Report will be sent on the corresponding monitor requests.
- Cancel button — If a call is on hold at the attendant and the attendant presses the start button, putting the previous call on hold, and then either dials a number and then presses the cancel button or presses the cancel button right away, the call that was originally put on hold will be reconnected and a Retrieved Event Report will be sent to the monitor request on the call.
- Release button — If only one call is active and the attendant presses the release button, the call will be dropped and the Connection Cleared Event Report will be sent to the monitor request on the call. If two calls are active at the attendant and the attendant then presses the release button, the calls will be transferred away from the attendant and a Transferred Event Report will be sent to the monitor request on the calls.
- Split button — If two calls are active at the attendant and the attendant presses the split button, the calls will be conferenced at the attendant and a Conferenced Event Report will be sent to the monitor requests monitoring the calls.

Attendant Auto-Manual Splitting

If an individual attendant receives a call with `cstaMonitorDevice` requests, then activates the Attendant Auto-Manual Splitting feature, a Held Event Report is returned to the monitor requests. The next event report sent depends on which button the attendant presses on the set (CANCEL = Retrieved, SPLIT = Conferenced, RELEASE = Transferred).

Attendant Call Waiting

Calls that provide event reports over `cstaMonitorDevice` requests and are extended by an attendant to a local, busy, single-line voice terminal will generate the following event reports:

- Held when the incoming call is split away by the attendant.
- Established when the attendant returns to the call.

The following events are generated, if the busy station does not accept the extended call and its returns:

- Delivered when the call is returned to the attendant.
- Established when the attendant returns to the call.

Attendant Control of Trunk Group Access

Calls that provide event reports over `cstaMonitorDevice` requests can access any trunk group controlled by the attendant. The attendant is alerted and places the call to its destination.

AUDIX

Calls that cover AUDIX do not maintain a simulated bridge appearance on the principal's station. The principal receives audible alerting followed by an interval of coverage response followed by the call dropping from the principal's set. When the principal receives alerting, the Delivered Event Report is sent. When the call is dropped from the principal's set because the call went to AUDIX coverage, the Diverted Event Report is sent.

Automatic Call Distribution (ACD)

Announcements

Announcements played while a monitored call is in a split queue, or as a result of an announcement vector command, create no event reports. Calls made directly to announcement extensions will have the same event report sent to the application as calls made to station extensions. In either case, no Queued Event Report is sent to the application.

Interflow

This occurs when a split redirects all calls to another split on another PBX by activating off-premise call forwarding.

When a monitored call interflows, event reports will cease except for the Network Reached (for non-PRI trunk) and trunk Connection Cleared Event Reports.

Night Service

The Delivered Event Report is sent when a call that is not being monitored enters an ACD split (not adjunct-controlled) with monitor requests and also has night service active.

Service Observing

A monitored call can be service observed provided that service observing is originated from a voice terminal and the service observing criteria is met. An Established Event Report is generated every time service observing is activated for a monitored call. A Connection Cleared Event Report is generated when the observer disconnects from the call.

For a `cstaMakeCall` call, the observer is bridged on the connection when the destination answers. When the destination is a trunk with answer supervision (includes PRI), the observer is bridged on when an actual far-end answer occurs. When the destination is a trunk without answer supervision, the observer is bridged on after the Network Reached (timeout) event.

Applicable events are “Established” (when the observer is bridged on) with the observer’s extension and “Connection Cleared” when the observer drops from the call. In addition, the observer may manipulate the call via Call Control requests to the same extent as he or she can via the station set.

Auto-Available Split

An auto-available split can be monitored as an ACD split and members of auto-available splits (agents) can be monitored as stations.

Bridged Call Appearance

A `cstaMonitorDevice` monitored station can have a bridged appearance(s) of its primary extension number appear at other stations. For bridging, event reports are provided based on the internal state of bridging parties with respect to the call. A call to the primary extension number will alert both the principal and the bridged appearance. Two or more Delivered Event Reports get triggered, one for the principal, and one for each of the bridged appearances. Two or more Established Event Reports may be triggered, if both the primary extension number and the bridged appearance(s) pick up the call. When the principal or bridging user goes on hook but the bridge itself does not drop from the call, no event report is sent but the state of that party changes from the connected state to the bridged state. When the principal or bridging user reconnects, another Established Event Report will be sent. A Connection Cleared Event Report will be triggered for the principal and each bridged appearance when the entire bridge drops from the call.

Members that are not connected to the call while the call is connected to another bridge member are in the “bridged” state. When the only connected member of the bridge transitions to the held state, the state for all members of the bridge changes to the held state even if they were previously in the bridged state. There is no event report sent to the bridged user monitor request for this transition.

Both the principal and bridging users may be individually monitored by a `cstaMonitorDevice`. Each will receive appropriate events as applicable to the monitored station. However, event reporting for a member of the bridge in the held state will be dependent on whether the transition was from the connected state or the bridged state.

CSTA Conference Call, Drop Call, Hold Call, Retrieve Call, and Transfer Call services are not permitted for parties in the bridged state and may also be more restrictive if the principal of the bridge has an analog set or if the exclusion option is in effect from a station associated with the bridge.

A CSTA Make Call request will always originate at the primary extension number of a user having a bridged appearance. For a call to originate at the bridged call appearance of a primary extension, that user must be off hook at that bridged appearance at the time the request is received.

⇒ NOTE:

A principal station with bridged call appearance can be single step conferenced into a call. Stations with bridged call appearance to the principal have the same bridged call appearance behavior, that is, if monitored, the station will receive Established And Conferenced Events when it joins the call. The station will not receive a Delivered Event.

Busy Verification of Terminals

A `cstaMonitorDevice`-monitored station may be busy-verified. An Established Event Report is provided when the verifying user is bridged in on a connection in which there is a `cstaMonitorDevice`-monitored station.

Call Coverage

If a call that goes to coverage is monitored by a monitor request on an ACD split or a VDN, the monitor request will receive the Delivered and Established Event reports.

For an alternate answering position that is monitored by a `cstaMonitorDevice` request, the Delivered and Established Event Reports are returned to its `cstaMonitorDevice` request.

The Diverted Event Report is sent to the principal’s `cstaMonitorDevice` request when an analog principal’s call goes to coverage. The Connection Cleared Event

Report is sent for the coverage station's monitor requests when the call that had been alerting at both the principal and the coverage is picked up at the principal.

Call Coverage Path Containing VDNs

When a call is diverted to a station/split coverage path and the coverage path is a VDN, the switch will provide the following event reports for the call:

- **Diverted Event Report** — This event report is sent to a monitor request on a station. A Diverted Event Report can also be sent to the diverted-from VDN's monitor request on the call, if the diverted-to VDN in the coverage path has a monitor request. The diverted-to VDN's monitor request receives a Delivered (to an ACD device) Event Report. If the diverted-to VDN in the coverage path has no active monitor request (not monitored), then no Diverted Event Report is sent to the diverted-from VDN's monitor request for the call.
- **Delivered (to ACD device) Event Report** — This report is only sent if the diverted-to VDN in the call coverage path has a monitor request.

All other event reports associated with calls in a VDN (for example, Queued and Delivered Event Reports) are provided to all monitor requests on the call.

Call Forwarding All Calls

No Diverted Event Report will be sent to a `cstaMonitorDevice` request for the forwarding station, since the call does not alert the extension that has Call Forwarding activated. This is only if the call was placed directly to "forwarded to station."

If a monitored call is forwarded off-PBX over a non-PRI facility, the Network Reached Event Report will be generated.

Call Park

A `cstaMonitorDevice`-monitored station can activate Call Park.

A call may be parked manually at a station by use of the "call park" button (with or without the conference and/or transfer buttons), or by use of the feature access code and the conference and/or transfer buttons.

When a call is parked by using the "call park" button without either the conference or the transfer buttons, there are no event reports generated. When the conference or transfer buttons are used to park a call, the Conferenced or Transferred Event Reports are generated. In this case, the "calling" and the "called" number in the Conferenced or Transferred Event Reports will be the same as that of the station on which the call was parked.

When the call is unparked, an Established Event Report is generated with the “calling” and “called” numbers indicating the station on which the call had been parked, and the “connected” number is that of the station unparking the call.

Call Pickup

A call alerting at a cstaMonitorDevice-monitored station may be picked up using Call Pickup. The station picking up (either the principal or the pickup user or both) may be monitored. An Established Event Report is sent to all monitor requests on the call when this feature is used. When a pickup user picks up the principal’s call, the principal’s set (if multifunction) retains a simulated bridge appearance and is able to connect to the call at any time. No event report is sent for the principal unless the principal connects in the call.

When a call has been queued first and then picked up by a pickup user, it is possible for a client application to see an Established Event Report without having seen any prior Delivered Event Reports.

Call Vectoring

A VDN can have a monitor request. Interactions between event reporting and call vectoring are shown in the following table.

Table 9-2. Interactions Between Feedback and Call Vectoring

Vector Step or Command	Event Report	When Sent	Cause
Vector Initialization	Delivered 88 ¹ (to ACD device)	encountered	
Queue to Main	— Queued — Failed	— successfully queues — queue full, no agents logged in	— queue full
Check Backup	— Queued — Failed	— successfully queues — queue full, no agents logged in	— queue full
Messaging Split	— Queued — Failed	— successfully queues — queue full, no agents logged in	— queue full
Announcement	none		

Table 9-2. Interactions Between Feedback and Call Vectoring

Vector Step or Command	Event Report	When Sent	Cause
Wait	none		
GoTo	none		
Stop	none		
Busy	— Failed	— Encountered	— busy
Disconnect	— Connection Cleared	— Facility Dropped	— busy
Go To Vector	none		
Route to (internal)	Delivered (to station device)		
Route To (external)	Network Reached		
Adjunct Routing	route		
Collected Digits	none		
Route To Digits (internal)	Delivered (to station device)		
Route To Digits (external)	Network Reached		
Converse Vector Command	<ul style="list-style-type: none"> — Queued Event — Delivered Event — Established Event — Connection Cleared Event 	<ul style="list-style-type: none"> — If the call queues for the agent or automated attendant (VRU) — When the call is delivered to an agent or the automated attendant — When the call is answered by the agent or automated attendant — When the call disconnects from the agent or automated attendant 	

1. Only reported over a VCN/ACD split monitor association.

Call Prompting

Up to 16 digits collected from the last "collect digit" vector command will be passed to the application in the Delivered Event Report³.

Lookahead Interflow

This feature is activated by encountering a "route to" vector command, with the route to destination being an off PBX number, and having the ISDN-PRI, Vectoring (Basic), and Lookahead Interflow options enabled on the Customer Options form.

For the originating PBX, the interactions are the same as with any call being routed to an off-PBX destination by the "route to" vector command.

For the receiving PBX, the lookahead interflow information element is passed in the ISDN message and will be included in all subsequent Delivered (to ACD device) Event Report⁴ for the call, when the information exists, and when the call is monitored.

Multiple Split Queueing

A Queued Event Report is sent for each split that the call queues to. Therefore, multiple call queued events could be sent to a client application for one call.

If a call is in multiple queues and abandons (caller drops), one Connection Cleared Event Report (cause normal) will be returned to the application followed by a Call Cleared Event Report.

When the call is answered at a split, the call will be removed from the other split's queue. No other event reports for the queues will be provided in addition to the Delivered and Established Event Reports.

Call Waiting

When an analog station is administered with this feature and a call comes in while the

user is busy on another call, the Delivered Event Report is sent to the client application.

-
3. The collected digits are sent in the private data.
 4. Lookahead Interflow Information is supported in private data.

Conference

Report is Manual conference from a cstaMonitorDevice monitored station is allowed, subject to the feature's restrictions. The Held Event Report is provided as a result of the first button push or first switch-hook flash. The Conferenced Event Report is provided as a result of the second button push or second switch-hook flash, and only if the conference is successfully completed. On a manual conference or on a Conference Call Service request, the Conferenced Event is sent to all the monitor requests for the resultant call.

Consult

When the covering user presses the Conference or Transfer feature button and receives a dial tone, a Held Event Report is returned to monitor requests of the call. A Service Initiated Event Report is then returned to the monitor requests on the covering user. After the Consult button is pressed by the covering user, Delivered and Established Event Reports are returned to monitor requests on the principal and covering user. Then the covering user can conference or transfer the call.

CTI Link Failure

When the connectivity of the CTI link between the G3 PBX and the G3PD is interrupted or reset, information of all calls received by the G3PD before are not reliable. When CTI link failure happens, all call records are destroyed and information such as User To User Info, User Entered Code are deleted from the G3PD. If the link is restored in time, the call events may resume for the new monitor requests (note that when CTI link is re-initialized, all monitor associations are aborted), but the Original Call Information for calls that exist before the link went down are not available.

Data Calls

Analog ports equipped with modems can be monitored by the cstaMonitorDevice Service and calls to and from ports can be monitored. However, Call Control Service requests may cause the call to be dropped by the modem.

DCS

With respect to event reporting, calls made over a DCS network are treated as off-PBX calls and only the Service Initiated, Network Reached, Call Cleared, and/or Connection Cleared Event Reports are generated. DCS/UDP extensions that are local to the PBX are treated as on-PBX stations. DCS/UDP extensions connected to the remote nodes are treated as off-PBX numbers.

Incoming DCS calls will provide a calling party number.

Direct Agent Calling and Number of Calls In Queue

Direct-agent calls will not be included in the calculation of number of calls queued for the Queued Event Report.

Drop Button Operation

The operation of this button is not changed with G3 CSTA Services.

When the "Drop" button is pushed by one party in a two-party call, the Connection Cleared Event Report is sent with the extension of the party that pushed the button. The originating party receives dial tone and the Service Initiated Event Report is reported on its cstaMonitorDevice requests.

When the "Drop" button is pushed by the controlling party in a conference, the Connection Cleared Event Report is sent with the extension of the party who was dropped off the call. This might be a station extension or a group extension. A group extension is provided in situations when the last added party to a conference was a group (for example, TEG, split, announcement, etc.) and the "Drop" button was used while the group extension was still alerting (or was busy). Since the controlling party does not receive dial tone (it is still connected to the conference), no Service Initiated Event Report is reported in this case.

Expert Agent Selection (EAS)

Logical Agents

Whenever logical agents are part of a monitored call, the following additional rules apply to the event reports:

- The callingDevice always contains the logical agent's physical station number (extension), even though a Make Call request might have contained a logical agent's login ID as the originating number (callingDevice).

- The answeringDevice and alertingDevice contain the logical agent's station extension and never contain the login ID. This is true regardless of whether the call was routed through a skill hunt group, whether the connected station has a logical agent currently logged in, or whether the call is an application-initiated or voice terminal-initiated direct agent call.
- The calledDevice contains the number that was dialed, regardless of the station connected to the call. For example, a call may be alerting an agent station, but the dialed number might have been a logical agent's login ID, a VDN, or another station.
- The Conferenced and Transferred Event Reports are an exception to this rule. In these events the addedParty contains the station extension of the transferred to or conferenced party when a local extension is involved. When an external extension is involved, the addedParty is unknown. If the transferred to or conferenced party is a hunt group or login ID and the call has not been delivered to a station, the addedParty contains the hunt group or login ID extension. If the call has been delivered to a station, the addedParty contains the station extension connected to the call.
- The alertingDevice in the Delivered and the queue in the Queued Event Report for logical direct agent calls contains a skill hunt group from the set of skills associated with the logical agent. Note that the skill hunt group is provided, even though an application-initiated, logical direct agent call request did not contain a skill hunt group.

Hold

Manually holding a call (either by using the Hold, Conference, Transfer buttons, or switch-hook flash) results in the Held Event Report being sent to all monitor requests for this call, including the held device. A held party is considered on the call for the purpose of receiving events relevant to that call.

Integrated Services Digital Network (ISDN)

The Make Call calls will follow Integrated Services Digital Network (ISDN) rules for the originator's name and number. The Service Initiated Event Report will not be sent for en-bloc BRI sets.

Multiple Split Queuing

When a call is queued in multiple ACD splits and then removed from the queue, the Delivered Event Report will provide the split extension of the alerting agent. There will be no other events provided for the splits from which the call was removed.

Personal Central Office Line (PCOL)

Members of a Personal Central Office Line (PCOL) may be monitored by the `cstaMonitorDevice` Service. PCOL behaves like bridging for the purpose of event reporting. When a call is placed to a PCOL group, the Delivered Event Report is provided to each member's `cstaMonitorDevice` requests. The `calledDevice` information passed in the Delivered event will be the default station characters. When one of the members answers the incoming call, the Established Event Report provides the extension of the station that answered the call. If another member connects to the call, another Established Event Report is provided. When a member goes on hook but the PCOL itself does not drop from the call, no event is sent but the state of that party changes from the connected state to the bridged state. The Connection Cleared Event Report is not sent to each member's `cstaMonitorDevice` requests until the entire PCOL drops from the call (as opposed to an individual member going on-hook). Members that are not connected to the call while the call is connected to another PCOL member are in the bridged state. When the only connected member of the PCOL transitions to the held state, the state for all members of the PCOL changes to the held state even if they were previously in bridged state. There is no event report sent to any `cstaMonitorDevice` request(s) for bridged users for this transition.

All members of the PCOL may be individually monitored by the `cstaMonitorDevice` Service. Each will receive appropriate events as applicable.

Primary Rate Interface (PRI)

Primary Rate Interface (PRI) facilities may be used for either inbound or outbound application monitored calls.

An incoming call over a PRI facility will provide the `callingDevice` and `calledDevice` information (CPN/BN/DNIS) which is passed on to the application in the Delivered (to ACD device) and Established Event Reports.

An outgoing call over a PRI facility provides call feedback events from the network.

A `cstaMakePredictiveCall` call will always use a call classifier on PRI facilities, whether the call is interworked or not. Although these facilities are expected to report call outcomes on the "D" channel, often interworking causes loss or delay of such reports. Progress messages reporting "busy," SITs, "alert," and "drop/disconnect" will cause the corresponding event report to be sent to the application. For `cstaMakePredictiveCall` calls, the "connected" number is interpreted as "far end answer" and is reported to the application as the Established Event Report when received before the call classifiers' "answer" indication. When received after the call classifier has reported an outcome, it will not be acted upon. A monitored outbound call over PRI facilities may generate the Delivered, Established, Connection Cleared, and/or Call Cleared Event Reports, if such a call goes ISDN end-to-end. If such a call interworks, the ISDN

PROGress message is mapped into a Network Reached Event Report. In this case, only the Connection Cleared or Call Cleared Event Reports may follow.

Ringback Queuing

CstaMakePredictiveCall calls will be allowed to queue on busy trunks or stations.

When activated, the callback call will report events on the same callID as the original call.

Send All Calls (SAC)

For incoming calls, the Delivered Event Report is sent only for multifunction sets receiving calls while having SAC activated. The Delivered Event Report is not generated for analog sets when the SAC feature is activated and the set is receiving a call.

Service-Observing

CstaMonitorDevice monitored stations may be service-observed and observers. When a monitored station is the observer, and it is bridged onto a call for the purpose of service observing, the Established Event Report is sent to the observer's cstaMonitorDevice requests for as well as to all other monitor requests for that call.

Temporary Bridged Appearances

The operation of this feature has not changed with G3 CSTA Services. There is no event provided when a temporary bridged appearance is created at a multifunction set. If the user is connected to the call (becomes active on such an appearance), the Established Event Report is provided. If a user goes on hook after having been connected on such an appearance, a Connection Cleared Event Report (normal clearing) is generated for the disconnected extension (bridged appearance).

If the call is dropped from the temporary bridged appearance by someone else, a Connection Cleared Event Report is also provided.

Temporary bridged appearances are not supported with analog sets. Analog sets get the Diverted Event Report when such an appearance would normally be created for a multifunction set.

The call state provided to queries about extensions with temporary bridged appearances will be "bridged" if the extension is not active on the call or it will be "connected" if the extension is active on the call.

Terminating Extension Group (TEG)

Members of a TEG may be monitored by the cstaMonitorDevice Service. A TEG behaves similarly to bridging for the purpose of event reporting. If cstaMonitorDevice monitored stations are members of a terminating group, an incoming call to the group will cause a Delivered Event Report to be sent to all cstaMonitorDevice requests for members of the terminating group. On the cstaMonitorDevice request for the member of the group that answers the call, an Established Event Report is returned to the answering member's cstaMonitorDevice request(s) which contains the station that answered the call. All the cstaMonitorDevice requests for the other group members (nonanswering members without TEG buttons) receive a Diverted Event Report. When a button TEG member goes on hook but the TEG itself does not drop from the call, no event is sent but the state of that party changes from the connected state to the bridged state.

The Connection Cleared Event Report is not sent to each member's cstaMonitorDevice requests until the entire TEG drops from the call (as opposed to an individual member going on hook).

Members that are not connected to the call while the call is connected to another TEG member are in the bridged state. When the only connected member of the TEG transitions to the held state, the state for all members of the TEG changes to the held state even if they were previously in the bridged state. There is no event report sent over the cstaMonitorDevice requests for the bridged user(s) for this transition.

All members of the TEG may have individual cstaMonitorDevice request. Each will receive appropriate events as applicable to the monitored station.

Transfer

Manual transfer from a station monitored by a cstaMonitorDevice request is allowed subject to the feature's restrictions. The Held Event Report is provided as a result of the first button push (or switch-hook flash for analog sets). The Transferred Event Report is provided as a result of the second button push (or on-hook for analog sets), and only if the transfer is successfully completed. The Transferred Event Report is sent to all monitor requests for the resultant call.

Trunk-to-Trunk Transfer

Existing rules for trunk-to-trunk transfer from a station user will remain unchanged for monitored calls. In such cases, transfers requested via Transfer Call request will be negatively acknowledged. When this feature is enabled, monitored calls transferred from trunk-to-trunk will be allowed, but there will be no further notification.

Overview

The Routing Services are Computing Function Services. The services in this group allow the switch to request and receive routing instructions for a call. These instructions, issued by a client routing server application, are based upon the incoming call information provided by the switch.

The following Routing Services are available:

- Route End Event
- Route End Service (TSAPI Version 2)
- Route End Service (TSAPI Version 1)
- Route Register Abort Event
- Route Register Cancel Service
- Route Register Service
- Route Request Service (TSAPI Version 2)
- Route Request Service (TSAPI Version 1)
- Route Select Service (TSAPI Version 2)
- Route Select Service (TSAPI Version 1)
- Route Used Event (TSAPI Version 2)
- Route Used Event (TSAPI Version 1)

Route End Event

Direction: Switch to Client

Event: CSTARouteEndEvent

Service Parameters: routeRegisterReqID, routingCrossRefID, errorValue

Functional Description:

This event is sent by the switch to terminate a routing dialog for a call and to inform the routing server application of the outcome of the call routing.

Service Parameters:

<i>routeRegisterReqID</i>	[mandatory] Contains the handle to the routing registration session for which the application is providing routing services. The application received this handle in a <i>CSTARouteRegisterReqConfEvent</i> confirmation to a <i>cstaRouteRegisterReq()</i> request.
<i>routingCrossRefID</i>	[mandatory] Contains the handle to the CSTA call routing dialog for a call. The application previously received this handle in the <i>CSTARouteRequestExtEvent</i> for the call. This is the routing dialog that the switch is ending.
<i>errorValue</i>	[mandatory] Contains the cause code for the reason why the switch is ending the routing dialog. One of the following values will be returned: <ul style="list-style-type: none">■ GENERIC_UNSPECIFIED (0) (CS0/16) The call has been routed successfully.■ INVALID_CALLING_DEVICE (5) (CS3/15) Upon routing to an agent (for a direct-agent call), the agent is not logged in.■ PRIVILEGE_VIOLATION_ON_SPECIFIED_DEVICE (8) (CS3/43) Lack of calling permission; for example, for an ARS call, there is an insufficient Facility Restriction Level (FRL). For a direct-agent call, the originator's Class Of Restriction (COR) or the destination agent's COR does not allow a direct-agent call.■ INVALID_DESTINATION (14) (CS0/28) The destination address in the <i>cstaRouteSelectInv()</i> is invalid.■ INVALID_OBJECT_TYPE (18) (CS3/11) Upon routing to an agent (for direct-agent call), the agent is not a member of the specified split.■ INVALID_OBJECT_STATE (22) A Route Select request was received by G3PD in wrong state. A second Route Select request sent by the application before the routing dialog is ended may cause this.■ NO_ACTIVE_CALL (24) (CS0/86, CS3/86) The call was dropped (for example, caller abandons, vector disconnect timer times out, a non-queued call encounters a "stop" step, or the application clears the call) while waiting for a <i>cstaRouteSelectInv()</i> response.

- **NO_CALL_TO_ANSWER (28) (CS3/30)** The call has been redirected. The switch has canceled or terminated any outstanding `CSTARouteRequestExtEvent (s)` for the call after receiving the first valid `cstaRouteSelectInv()` message. The switch sends a Route End Event with this cause to all other outstanding `CSTARouteRequestExtEvent (s)` for the call. Note that this error can happen when Route Registers are registered for the same routing device from two different Tservers and the switch is set to send multiple Route Requests for the same call.
- **RESOURCE_BUSY (33) (CS0/17)** The destination is busy and does not have coverage. The caller will hear either a reorder or busy tone.
- **PERFORMANCE_LIMIT_EXCEEDED (52) (CS0/102)** Call vector processing encounters any steps other than wait, announcement, goto, or stop after the `CSTARouteRequestExtEvent (adjunct routing command)` has been issued. This can also happen when a wait step times out. When the switch sends `CSTARouteEndEvent` with this cause, call vector processing continues.

Detailed Information:

An application may receive one Route End Event and one Universal Failure for a Route Select request for the same call in one of the following call scenarios:

- Switch/G3PD sends a Route Request to application.
- Caller drops the call.
- Application sends a Route Select Request to G3PD.
- Switch/G3PD sends a Route End Event (`errorValue = NO_ACTIVE_CALL`) to application.
- G3PD receives the Route Select Request, but call has been dropped.
- G3PD sends Universal Failure for the Route Select request (`errorValue = INVALID_CROSS_REF_ID`) to application.

Syntax

```

#include <acs.h>
#include <csta.h>

// CSTARouteEndEvent - Route Select Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTAEVENTREPORT
    EventType_t eventType; // CSTA_ROUTE_END
} ACSEventHeader_t;

typedef struct CSTARouteEndEvent_t {
    RouteRegisterReqID_t routeRegisterReqID,
    RoutingCrossRefID_t routingCrossRefID,
    CSTAUniversalFailure_t errorValue,
} CSTARouteEndEvent_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            union
            {
                CSTARouteEndEvent_t routeEnd;
            } u;
        } cstaEventReport;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

```

Route End Service (TSAPI Version 2)

Direction: Client to Switch

Function: cstaRouteEndInv()

Service Parameters: routeRegisterReqID, routingCrossRefID, errorValue

Ack Parameters: noData

Nak Parameter: universalFailure

Functional Description:

This service is sent by the routing server application to terminate a routing dialog for a call. The service request includes a cause value giving the reason for the routing dialog termination.

Service Parameters:

routeRegisterReqID [mandatory] Contains the handle to the routing registration session for which the application is providing routing services. The routing server application received this handle in a `CSTARouteRegisterReqConfEvent` confirmation to a `cstaRouteRegisterReq()` request.

routingCrossRefID [mandatory] Contains the handle to the CSTA call routing dialog for a call. The routing server application previously received this handle in the `CSTARouteRequestExtEvent` for the call. This is the routing dialog that the application is terminating.

errorValue [mandatory] Contains the cause code for the reason why the application is terminating the routing dialog. One of the following values can be sent:

- Any CSTA `universalFailure` error code

The `errorValue` is ignored by the G3 switch and has no effect for the routed call, but it must be present in the API. Suggested error codes that may be useful for error logging purposes are:

- `GENERIC_UNSPECIFIED` (0) Normal termination (for example, application does not want to route the call or does not know how to route the call).
- `INVALID_CSTA_DEVICE_IDENTIFIER` (12) An invalid `routeRegisterReqID` has been specified in the `RouteEndInv()` request.
- `RESOURCE_BUSY` (33) Routing server is too busy to handle the route request.
- `RESOURCE_OUT_OF_SERVICE` (34) Routing service temporarily unavailable due to internal problem (for example, the database is out of service).

Ack Parameters:

noData None for this service.

Nak Parameter:

- universalFailure*** If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error value, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:
- INVALID_CROSS_REF_ID (17) An invalid routeRegisterReqID or routeCrossRefID has been specified in the Route Ended request.

Detailed Information:

- If an application terminates a Route Request via a cstaRouteEndInv(), the switch continues vector processing.
- An application may receive one Route End Event and one Universal Failure for a cstaRouteEndInv() request for the same call in the following call scenario:
 - Switch/G3PD sends a CSTARouteRequestEvent to application.
 - Caller drops the call.
 - Application sends a cstaRouteEndInv() request to G3PD.
 - Switch/G3PD sends a CSTARouteEndEvent (errorValue = NO_ACTIVE_CALL) to application.
 - G3PD receives the cstaRouteEndInv() request, but call has been dropped.
 - G3PD sends universalFailure for the cstaRouteEndInv() request (errorValue = INVALID_CROSS_REF_ID) to application.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaRouteEndInv() - Service Request

RetCode_t  cstaRouteEndInv (
    ACSHandle_t      acsHandle,
    InvokeID_t       invokeID,
    RouteRegisterReqID_t  routeRegisterReqID,
    RoutingCrossRefID_t  routingCrossRefID,
    CSTAUniversalFailure_t  errorValue,
    PrivateData_t     *privateData);

typedef long        RouteRegisterReqID_t;

typedef long        RoutingCrossRefID_t;
```

Route End Service (TSAPI Version 1)

Direction: Client to Switch

Function: cstaRouteEnd()

Service Parameters: routeRegisterReqID, routingCrossRefID, errorValue

Functional Description:

This service is sent by the routing server application to terminate a routing dialog for a call. The service request includes a cause value giving the reason for the routing dialog termination.

Detailed Information:

An application may receive two CSTARouteEndEvent(s) for the same call in one of the following call scenarios:

- Switch/G3PD sends a CSTARouteRequestEvent to application.
- Caller drops the call.
- Application sends a cstaRouteSelect() to G3PD.
- Switch/G3PD sends a CSTARouteEndEvent (errorValue = NO_ACTIVE_CALL) to application.
- G3PD receives the cstaRouteSelect() Request, but call has been dropped.
- G3PD sends CSTARouteEndEvent (errorValue = INVALID_CROSS_REF_ID) to application.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaRouteEnd() - Service Request

RetCode_t  cstaRouteEnd (
    ACSHandle_t      acsHandle,
    RouteRegisterReqID_t  routeRegisterReqID,
    RoutingCrossRefID_t  routingCrossRefID,
    CSTAUniversalFailure_t  errorValue,
    PrivateData_t      *privateData);

typedef long      RouteRegisterReqID_t;

typedef long      RoutingCrossRefID_t;
```

Route Register Abort Event

Direction: Switch to Client
Event: CSTARouteRegisterAbortEvent
Service Parameters: routeRegisterReqID

Functional Description:

This event notifies the application that the G3PD or switch aborted a routing registration session. After the abort occurs, the application receives no more `CSTARouteRequestExtEvent(s)` from this routing registration session and the `routeRegisterReqID` is no longer valid. The routing requests coming from the routing device will be sent to the default routing server, if a default routing registration is still active.

Service Parameters:

routeRegisterReqID [mandatory] Contains the handle to the routing registration session for which the application is providing routing services. The application received this handle in a `CSTARouteRegisterReqConfEvent` confirmation to a `cstaRouteRegisterReq()` request.

Detailed Information:

- If no CTI link has ever received any `CSTARouteRequestExtEvent(s)` for the registered routing device and all of the CTI links are down, this event is not sent.
- In a multi-link configuration, if at least one link that has received at least one `CSTARouteRequestExtEvent` for the registered routing device is up, this event is not sent. It is sent only when all of the CTI links that have received at least one `CSTARouteRequestExtEvent` for the registered routing device are down.

⇒ NOTE:

How the G3 switch sends the `CSTARouteRequestExtEvent(s)` for the registered routing device, via which CTI links, is controlled by the call vectoring administered on the switch. A routing device can receive `CSTARouteRequestExtEvent(s)` from different CTI links. It is possible that links are up and down without generating this event.

- If the application wants to continue the routing service after the CTI link is up, it must issue a `cstaRouteRegisterReq()` to re-establish a routing registration session for the routing device.

Syntax

```
#include <acs.h>
#include <csta.h>

// CSTARouteRegisterAbortEvent

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTAEVENTREPORT
    EventType_t eventType; // CSTA_ROUTE_REGISTER_ABORT
} ACSEventHeader_t;

typedef struct CSTARouteRegisterAbortEvent_t {
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            union
            {
                CSTARouteRegisterAbortEvent_t routeCancel;
            } u;
        } cstaEventReport;
    } event;
} CSTAEvent_t;

typedef struct CSTARouteRegisterAbortEvent_t {
    RouteRegisterReqID_t routeRegisterReqID,
} CSTARouteRegisterAbortEvent_t;

typedef long RouteRegisterReqID_t;
```

Route Register Cancel Service

Direction: Client to Switch

Function: `cstaRouteRegisterCancel()`

Confirmation Event: `CSTARouteRegisterCancelConfEvent`

Service Parameters: `routeRegisterReqID`

Ack Parameters: `noData`

Nak Parameter: `universalFailure`

Functional Description:

Client applications use `cstaRouteRegisterCancel()` to cancel a previously registered `cstaRouteRegisterReq()` session. When this service request is positively acknowledged, the client application is no longer a routing server for the specific routing device and the G3PD stops sending `CSTARoutingRequestEvent(s)` from the specific routing device associated with the `routeRegisterReqID` to the requesting client application. The G3PD will send any further `CSTARoutingRequestEvent(s)` from the routing device to the default routing server application, if there is one registered.

Service Parameters:

routeRegisterReqID [mandatory] Contains the handle to the routing registration session for which the application is canceling. The routing server application received this handle in a `CSTARouteRegisterReqConfEvent` confirmation to a `cstaRouteRegisterReq()` request.

Ack Parameters:

noData None for this service.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a `CSTAUniversalFailureConfEvent`. The error parameter in this event may contain the following error value, or one of the error values described in the “`CSTAUniversalFailureConfEvent`” section in Chapter 3:

- `INVALID_CSTA_DEVICE_IDENTIFIER (12)` An invalid `routeRegisterReqID` has been specified in the request.

Detailed Information:

An application may receive `CSTARouteRequestExtEvent` after a `cstaRouteRegisterCancel` request is sent and before a `CSTARouteRegisterCancelConfEvent` response is received. The application should ignore the `CSTARouteRequestExtEvent`. If a `cstaRouteSelectInv()` request is sent for the `CSTARouteRequestExtEvent`, a `CSTARouteEndEvent` response will be received with error `INVALID_CSTA_DEVICE_IDENTIFIER`. If a `cstaRouteEndInv()` request is sent for the `CSTARouteRequestExtEvent`, it will be ignored. The outstanding `CSTARouteRequestExtEvent` will receive no response and will be timed out eventually.

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaRouteRegisterCancel() - Service Request

RetCode_t  cstaRouteRegisterCancel (
    ACSHandle_t      acsHandle,
    InvokeID_t       invokeID,
    RouteRegisterReqID_t  routeRegisterReqID,
    PrivateData_t    *privateData);

typedef long          RouteRegisterReqID_t;

// CSTARouteRegisterCancelConfEvent - Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTACONFIRMATION
    EventType_t eventType; // CSTA_ROUTE_REGISTER_CANCEL_CONF
} ACSEventHeader_t;

typedef struct CSTARouteRegisterCancelConfEvent_t {
    RouteRegisterReqID_t routeRegisterReqID;
} CSTARouteRegisterCancelConfEvent_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            InvokeID_t invokeID;
            union
            {
                CSTARouteRegisterCancelConfEvent_t routeCancel;
            } u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

```

Route Register Service

Direction: Client to Switch

Function: cstaRouteRegisterReq()

Service Parameters: routingDevice

Ack Parameters: routeRegisterReqID

Nak Parameter: universalFailure

Functional Description:

Client applications use cstaRouteRegisterReq() to register as a routing server for CSTARouteRequestExtEvent from a specific device. The application must register for routing services before it can receive any CSTARouteRequestExtEvent(s) from the routing device. An application may be a routing server for more than one routing device. However, for a specific routing device, the G3PD allows only one application registered as the routing server. If a routing device already has a routing server registered, subsequent cstaRouteRegisterReq() requests will be negatively acknowledged.

Service Parameters:

routingDevice [mandatory] Contains the device identifier of the routing device for which the application requests to be the routing server. A valid routing device on a G3 switch is a VDN extension which has the proper routing vector step set up to send the Route Requests to a G3PD. A NULL device identifier indicates that the requesting application will be the default routing server for the G3 switch. A default routing server will receive `CSTARouteRequestExtEvent(s)` from routing devices of the G3 switch that do not have a registered routing server.

Ack Parameters:

routeRegisterReqID [mandatory] Contains a handle to the routing registration session for a specific routing device (or for the default routing server). All routing dialogs (identified by `routingCrossRefID [s]`) for a routing device occur over this routing registration session.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a `CSTAUniversalFailureConfEvent`. The error parameter in this event may contain the following error value, or one of the error values described in the “`CSTAUniversalFailureConfEvent`” section in Chapter 3:

- `OUTSTANDING_REQUEST_LIMIT_EXCEEDED (44)`
The specified routing device already has a registered routing server.

Detailed Information:

- The `cstaRouteRegisterReq()` is handled by the G3PD, not by the G3 switch. The Route Requests are sent from the switch to the G3PD through call vector processing. From the perspective of the switch, the G3PD is the routing server. The G3PD processes the Route Requests and sends the `CSTARouteRequestExtEvent(s)` to the proper routing servers based on the route registrations from applications.
- If no routing server is registered for the G3 switch, all Route Requests from the switch will be terminated by the G3PD with a Route End Request, as if `cstaRouteEndInv()` requests were received from a routing server.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaRouteRegisterReq() - Service Request

RetCode_t  cstaRouteRegisterReq (
            ACSHandle_t          acsHandle,
            InvokeID_t          invokeID,
            DeviceID_t          *routingDevice,
            PrivateData_t       *privateData);

typedef long          RouteRegisterReqID_t;

// CSTARouteRegisterReqConfEvent - Service Response

typedef struct
{
            ACSHandle_t          acsHandle;
            EventClass_t        eventClass;
            EventType_t         eventType;
} ACSEventHeader_t;

typedef struct CSTARouteRegisterReqConfEvent_t {
            RouteRegisterReqID_t  registerReqID;
} CSTARouteRegisterReqConfEvent_t;

typedef struct
{
            ACSEventHeader_t eventHeader;
            union
            {
                    struct
                    {
                            InvokeID_t  invokeID;
                            union
                            {
                                    CSTARouteRegisterReqConfEvent_t routeRegister;
                            } u;
                    } cstaConfirmation;
            } event;
            char          heap[CSTA_MAX_HEAP];
} CSTAEvent_t;
```

Route Request Service (TSAPI Version 2)

Direction: Switch to Client

Event: CSTARouteRequestExtEvent

Private Data Event: ATTRouteRequestEvent (private data version 6),

ATTV5RouteRequestEvent (private data version 5),

ATTV4RouteRequestEvent (private data versions 2, 3, and 4)

Service Parameters: routeRegisterReqID, routingCrossRefID, currentRoute, callingDevice, routedCall, routedSelAlgorithm, priority, setupInformation

Private Parameters: trunkGroup, trunkMember, lookaheadInfo,

userEnteredCode, userInfo, ucid, callOriginatorInfo, flexibleBilling

Ack Parameters: N/A; see cstaRouteSelectInv() (“Route Select Service (TSAPI Version 2)”)

Nak Parameter: N/A; see cstaRouteEndInv() (“Route End Service (TSAPI Version 2)”)

Functional Description:

The switch sends a CSTARouteRequestExtEvent to request a destination for a call arrived on a routing device from a routing server application. The application may have registered as the routing server for the routing device on the switch that is making the request, or it may have registered as the default routing server. The CSTARouteRequestExtEvent includes call-related information. A routing server application typically uses the call-related information and a database to determine the destination for the call. The routing server application responds to the CSTARouteRequestExtEvent via a cstaRouteSelectInv() request that specifies a destination for the call or a cstaRouteEndInv() request, if the application has no destination for the call.

Service Parameters:

<i>routeRegisterReqID</i>	[mandatory] Contains a handle to the routing registration session for which the application is providing routing service. The routing server application received this handle in a CSTARouteRegisterReqConfEvent confirmation to a cstaRouteRegisterReq() request.
<i>routingCrossRefID</i>	[mandatory] Contains the handle for the routing dialog of this call. This identifier is unique within a routing session identified by the routeRegisterReqID.
<i>currentRoute</i>	[mandatory] Specifies the destination of the call. This is the VDN extension number first entered by the call (see "Detailed Information:").
<i>callingDevice</i>	[optional - supported] Specify the call origination device. This is the calling device number for on-PBX originators or incoming calls over PRI facilities. For incoming calls over non-PRI facilities, the trunk identifier ¹ is provided.
<i>routedCall</i>	[optional - supported] Specifies the callID of the call that is to be routed. This is the connectionID of the routed call at the routing device.
<i>routedSetAlgorithm</i>	[optional - partially supported] Indicates the type of routing algorithm requested. Type is set to SV_NORMAL.
<i>priority</i>	[optional - not supported] Indicates the priority of the call and may affect selection of alternative routes.
<i>setupInformation</i>	[optional - not supported] Contains an ISDN call setup message if available.

1. The trunk identifier is a dynamic device identifier. It cannot be used to access a trunk in the G3 switch

Private Parameters:

- trunkGroup*** [optional] Specifies the trunk group number from which the call is originated. The callingDevice and trunk parameters are mutually exclusive. Beginning with G3V8, both the calling device and trunk group may be present. Prior to G3V8, one or the other will be present, but not both. This parameter is supported by private data version 5 and later only.
- trunkMember*** [optional] This parameter is supported beginning with G3V4. It specifies the trunk member number from which this call originated.
- Beginning with G3V8, trunk member number is provided regardless of whether the calling device is available. Prior to G3V8, trunkMember number is provided only if the calling device is unavailable
- trunk*** [optional] Specifies the trunk group number from which the call is originated. The callingDevice and trunk parameters are mutually exclusive. One or the other will be present, but not both. This parameter is supported by private data versions 2, 3, and 4.
- lookaheadInfo*** [optional] Specifies the lookahead interflow information received from the incoming call that is to be routed. The lookahead interflow is a G3 switch feature that routes some of the incoming calls from one switch to another so that they can be handled more efficiently and will not be lost. The switch that overflows the call provides the lookahead interflow information. The routing server application may use the lookahead interflow information to determine the destination of the call. Please refer to the DEFINITY Generic 3 Feature Description for more information about lookahead interflow. If the lookahead interflow type is set to "LAI_NO_INTERFLOW", no lookahead interflow private data is provided with this event.
- userEnteredCode*** [optional] Specifies the code/digits that may have been entered by the caller through the G3 call prompting feature or the collected digits feature. If the userEnteredCode code is set to "UE_NONE", no userEnteredCode private data is provided with this event.

userInfo

[optional] Contains user-to-user information. This parameter allows the application to associate caller information, up to 32 or 96 bytes, with a call. It may be a customer number, credit card number, alphanumeric digits, or a binary string.

Prior to G3V8, the maximum length of *userInfo* was 32 bytes. Beginning with G3V8, the maximum length of *userInfo* was increased to 96 bytes.

⇒ NOTE:

An application using private data version 5 and earlier can only receive a maximum of 32-byte data for *userInfo*, regardless of the size data that is sent by the switch.

The following UUI protocol types are supported:

- UUI_NONE — There is no data provided in the data parameter.
- UUI_USER_SPECIFIC — The content of the data parameter is a binary string. The correct size (maximum of 32 or 96 bytes) of data must be specified in the size parameter.
- UUI_IA5_ASCII — The content of the data parameter must be a null-terminated IA5 (ASCII) character string. The correct size (maximum of 32 or 96 bytes excluding the null terminator) of data must be specified in the size parameter.

<i>ucid</i>	[optional] Specifies the Universal Call ID (UCID) of the routed call. The UCID is a unique call identifier across switches and the network. A valid UCID is a null-terminated ASCII character string. If there is no UCID associated with this call, the <i>ucid</i> contains the ATT_NULL_UCID (a 20-character string of all zeros). This parameter is supported by private data version 5 and later only.
<i>callOriginator</i>	[optional] Specifies the callOriginatorInfo of the call originator such as coin call, 800-service call, or cellular call. This information is from the network, not from the DEFINITY switch. The type is defined in Bellcore publication "Local Exchange Routing Guide" (document number TR-EOP-000085). A list of the currently defined codes, as of June 1994, is in the Detailed Information sub-section of the "Delivered Event" section. This parameter is supported by private data version 5 and later only.
<i>flexibleBilling</i>	[optional] Specifies whether the Flexible Billing feature is allowed for this call and the Flexible Billing customer option is assigned on the switch. If this parameter is set to TRUE, the billing rate can be changed for the incoming 900-type call using the Set Bill Rate Service. This parameter is supported by private data version 5 and later only.

Ack Parameters:

N/A See *cstaRouteSelectInv()*.

Nak Parameter:

N/A See *cstaRouteEndInv()*.

Detailed Information:

- The Routing Request Service can only be administered through the Basic Call Vectoring feature. The switch initiates the Routing Request when the Call Vectoring processing encounters the adjunct routing command in a call vector. The vector command will specify a CTI link's extension through which the switch will send the Route Request to the G3PD.
- Multiple adjunct routing commands are allowed in a call vector. In G3V3, the Multiple Outstanding Route Requests feature allows 16 outstanding Route Requests per call. The Route Requests can be over the same or different CTI links. The requests are all made from the same vector. They may be specified back-to-back, without intermediate (wait, announcement, goto, or stop) steps. If the adjunct routing commands are not specified back-to-back, pre-G3V3 adjunct routing functionality will apply. This means that previous outstanding Route Requests are canceled when an adjunct routing vector step is executed.

- The first Route Select response received by the switch is used as the route for the call, and all other Route Requests for the call are canceled via `CSTARouteEndEvent(s)`.
- If an application terminates the `CSTARouteRequestExtEvent` request via a `cstaRouteEndInv()`, the switch continues vector processing.
- A `CSTARouteRequestExtEvent` request will not affect the Call Event Reports.
- Like Delivered or Established Event, the Route Request `currentRoute` parameter contains the called device. In release 1 and release 2, the `currentRoute` in Route Request contains the originally called device if there is no distributing device, or the distributing device if the call vectoring with VDN override feature of the PBX is turned on. In the later case, the originally called device is not reported. The `distributingDevice` feature is not supported in the Route Request private data. See the "Delivered Event" section for detailed information on the `distributingDevice` parameter.

Syntax

```

#include <acs.h>
#include <csta.h>

// CSTARouteRequestExtEvent - CSTA Unsolicited Event

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTAREQUEST
    EventType_t eventType; // CSTA_ROUTE_REQUEST_EXT
} ACSEventHeader_t;

typedef long          RouteRegisterReqID_t;

typedef long          RoutingCrossRefID_t;

typedef char          DeviceID_t[64];

typedef struct ExtendedDeviceID_t {
    DeviceID_t      deviceID;
    DeviceIDType_t deviceIDType;
    DeviceIDStatus_t deviceIDStatus;
} ExtendedDeviceID_t;

typedef ExtendedDeviceID_t CallingDeviceID_t;

typedef ExtendedDeviceID_t CalledDeviceID_t;

typedef enum ConnectionID_Device_t {
    STATIC_ID = 0,
    DYNAMIC_ID = 1
} ConnectionID_Device_t;

typedef struct ConnectionID_t {
    long          callID;
    DeviceID_t    deviceID;
    ConnectionID_Device_t devIDType;
} ConnectionID_t;

typedef enum SelectValue_t {
    SV_NORMAL = 0,
    SV_LEAST_COST = 1,
    SV_EMERGENCY = 2,
    SV_ACD = 3,
    SV_USER_DEFINED = 4
} SelectValue_t;

```


Syntax (Continued)

```

typedef struct SetupValues_t {
    int                length;
    unsigned char     *value;
} SetupValues_t;

typedef struct CSTARouteRequestExtEvent_t {
    RouteRegisterReqID_t    routeRegisterReqID;
    RoutingCrossRefID_t    routingCrossRefID;

    CalledDeviceID_tcurrentRoute;// TSAPI V1 and V2 are
                                // different
    CallingDeviceID_tcallingDevice;// TSAPI V1 and V2 are
                                // different

    ConnectionID_t        routedCall;
    SelectValue_t         routedSelAlgorithm;
    Boolean                priority;
    SetupValues_t         setupInformation;
} CSTARouteRequestExtEvent_t;

typedef struct
{
    ACSEventHeader_teventHeader;
    union
    {
        struct
        {
            InvokeID_t    invokeID;// Unused for Route Request Event
                union
                {
                    CSTARouteRequestExtEvent_trouteRequestExt;
                } u;
        } cstaRequest;
    } event;
    char                heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

```

Private Data Version 6 Syntax

If private data accompanies a `CSTARouteRequestExtEvent`, then the private data would be stored in the location that the application specified as the `privateData` parameter in the `acsGetEventBlock()` or `acsGetEventPoll()` request. If the `privateData` pointer is set to `NULL` in these requests, then the `CSTARouteRequestExtEvent` does not deliver private data to the application.

If `acsGetEventBlock()` or `acsGetEventPoll()` returns a Private Data length of 0, then no private data is provided with this Route Request Event.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTRouteRequestEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventTypeeventType; // ATTV6_ROUTE_REQUEST
    union
    {
        ATTRouteRequestEvent_trouteRequest;
    } u;
} ATTEvent_t;

typedef struct ATTRouteRequestEvent_t
{
    DeviceID_t          trunkGroup;
    ATTLookaheadInfo_t  lookaheadInfo;
    ATTUserEnteredCode_t userEnteredCode;
    ATTUserToUserInfo_t userInfo;
    ATTUCID_t           ucid;
    ATTCallOriginatorInfo_t callOriginatorInfo;
    Boolean             flexibleBilling;
    DeviceID_t          trunkMember;
} ATTRouteRequestEvent_t;

typedef char ATTUCID_t[64];

typedef struct ATTLookaheadInfo_t
{
    ATTInterflow_t      type;
    ATTPriority_t       priority;
    short               hours;
    short               minutes;
    short               seconds;
    DeviceID_t          sourceVDN;
}
```

Private Data Version 6 Syntax (Continued)

```
        ATTUnicodeDeviceID_tuSourceVDN; // sourceVDN in Unicode
} ATTLookaheadInfo_t;
```

```
typedef enum ATTInterflow_t
{
    LAI_NO_INTERFLOW= -1,    // indicates Info not present
    LAI_ALL_INTERFLOW= 0,
    LAI_THRESHOLD_INTERFLOW= 1,
    LAI_VECTURING_INTERFLOW= 2
} ATTInterflow_t;
```

```
typedef enum ATTPriority_t
{
    LAI_NOT_IN_QUEUE= 0,
    LAI_LOW           = 1,
    LAI_MEDIUM        = 2,
    LAI_HIGH           = 3,
    LAI_TOP            = 4
} ATTPriority_t;
```

```
typedef struct ATTUserEnteredCode_t
{
    ATTUserEnteredCodeType_t      type;
    ATTUserEnteredCodeIndicator_t indicator;
    char                           data[25];
    DeviceID_t                      collectVDN;
} ATTUserEnteredCode_t;
```

```
typedef enum ATTUserEnteredCodeType_t
{
    UE_NONE           = -1,
    UE_ANY            = 0,
    UE_LOGIN_DIGITS   = 2,
    UE_CALL_PROMPTER = 5,
    UE_DATA_BASE_PROVIDED = 17,
    UE_TONE_DETECTOR  = 32
} ATTUserEnteredCodeType_t;
```

```
typedef enum ATTUserEnteredCodeIndicator_t
{
    UE_COLLECT = 0,
    UE_ENTERED = 1
} ATTUserEnteredCodeIndicator_t;
```

Private Data Version 6 Syntax (Continued)

```
#define ATT_MAX_USER_INFO 129
#define ATT_MAX_UII_SIZE 96
#define ATTV5_MAX_UII_SIZE 32

typedef struct ATTUserToUserInfo_t {
    ATTUIIProtocolType_t type;
    struct {
        short length; // 0 indicates UII not present
        unsigned char value[ATT_MAX_USER_INFO];
    } data;
} ATTUserToUserInfo_t;

typedef enum ATTUIIProtocolType_t
{
    UII_NONE = -1, // indicates not specified
    UII_USER_SPECIFIC= 0, // user-specific
    UII_IA5_ASCII = 4 // null terminated ascii char string
} ATTUIIProtocolType_t;

typedef struct ATTCallOriginatorInfo_t
{
    Boolean hasInfo; // if FALSE, no callOriginatorType
    short callOriginatorType;
} ATTCallOriginatorInfo_t;
```

Private Data Version 5 Syntax

If private data accompanies a CSTARouteRequestExtEvent, then the private data would be stored in the location that the application specified as the privateData parameter in the acsGetEventBlock() or acsGetEventPoll() request. If the privateData pointer is set to NULL in these requests, then the CSTARouteRequestExtEvent does not deliver private data to the application.

If acsGetEventBlock() or acsGetEventPoll() returns a Private Data length of 0, then no private data is provided with this Route Request Event.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV5RouteRequestEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventType eventType;    // ATTV5_ROUTE_REQUEST
    union
    {
        ATTV5RouteRequestEvent_tv5routeRequest;
    } u;
} ATTEvent_t;

typedef struct ATTV5RouteRequestEvent_t
{
    DeviceID_t                trunkGroup;
    ATTLookaheadInfo_t        lookaheadInfo;
    ATTUserEnteredCode_t      userEnteredCode;
    ATTUserToUserInfo_t       userInfo;
    ATTUCID_t                  ucid;
    ATTCallOriginatorInfo_t   callOriginatorInfo;
    Boolean                    flexibleBilling;
} ATTV5RouteRequestEvent_t;

typedef char ATTUCID_t[64];

typedef struct ATTLookaheadInfo_t
{
    ATTInterflow_t            type;
    ATTPriority_t             priority;
    short                     hours;
    short                     minutes;
    short                     seconds;
    DeviceID_t                 sourceVDN;
    ATTUnicodeDeviceID_tuSourceVDN; // sourceVDN in Unicode
} ATTLookaheadInfo_t;
```

Private Data Version 5 Syntax (Continued)

```
typedef enum ATTInterflow_t
{
    LAI_NO_INTERFLOW= -1,      // indicates Info not present
    LAI_ALL_INTERFLOW      = 0,
    LAI_THRESHOLD_INTERFLOW = 1,
    LAI_VECTURING_INTERFLOW = 2
} ATTInterflow_t;

typedef enum ATTPriority_t
{
    LAI_NOT_IN_QUEUE      = 0,
    LAI_LOW                = 1,
    LAI_MEDIUM            = 2,
    LAI_HIGH               = 3,
    LAI_TOP                = 4
} ATTPriority_t;

typedef struct ATTUserEnteredCode_t
{
    ATTUserEnteredCodeType_t      type;
    ATTUserEnteredCodeIndicator_t indicator;
    char                          data[25];
    DeviceID_t                    collectVDN;
} ATTUserEnteredCode_t;

typedef enum ATTUserEnteredCodeType_t
{
    UE_NONE                = -1,
    UE_ANY                  = 0,
    UE_LOGIN_DIGITS        = 2,
    UE_CALL_PROMPTER       = 5,
    UE_DATA_BASE_PROVIDED  = 17,
    UE_TONE_DETECTOR       = 32
} ATTUserEnteredCodeType_t;

typedef enum ATTUserEnteredCodeIndicator_t
{
    UE_COLLECT      = 0,
    UE_ENTERED      = 1
} ATTUserEnteredCodeIndicator_t;

typedef struct ATTV5UserToUserInfo_t
{
    ATTUIProtocolType_t type;
    struct {
        short      length;    // 0 indicates UUI not present
        unsigned char value[33];
    } data;
} ATTV5UserToUserInfo_t;
```

Private Data Version 5 Syntax (Continued)

```
typedef enum ATUUUIProtocolType_t
{
    UUI_NONE      = -1,          // indicates not specified
    UUI_USER_SPECIFIC= 0,       // user-specific
    UUI_IA5_ASCII= 4            // null terminated ascii char string
} ATUUUIProtocolType_t;

typedef struct ATTCallOriginatorInfo_t
{
    Boolean      hasInfo;       // if FALSE, no callOriginatorType
    short        callOriginatorType;
} ATTCallOriginatorInfo_t;
```

Private Data Versions 2-4 Syntax

```

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV4RouteRequestEvent - CSTA Unsolicited Event Private Data

typedef struct
{
    ATTEventTypeeventType; // ATTV4_ROUTE_REQUEST
    union
    {
        ATTV4RouteRequestEvent_tv4routeRequest;
    } u;
} ATTEvent_t;

typedef struct ATTV4RouteRequestEvent_t
{
    DeviceID_t            trunk;
    ATTV4LookaheadInfo_t lookaheadInfo
    ATTUserEnteredCode_t userEnteredCode;
    ATTUserToUserInfo_t  userInfo;
} ATTV4RouteRequestEvent_t;

typedef structATTV4LookaheadInfo_t
{
    ATTInterflow_t        type;
    ATTPriority_t         priority;
    short                 hours;
    short                 minutes;
    short                 seconds;
    DeviceID_t            sourceVDN;
} ATTLookaheadInfo_t;

typedef enum ATTInterflow_t
{
    LAI_NO_INTERFLOW= -1, // indicates Info not present
    LAI_ALL_INTERFLOW= 0,
    LAI_THRESHOLD_INTERFLOW= 1,
    LAI_VECTORING_INTERFLOW= 2
} ATTInterflow_t;

```


Private Data Versions 2-4 Syntax (Continued)

```

typedef enum ATTPriority_t
{
    LAI_NOT_IN_QUEUE      = 0,
    LAI_LOW                = 1,
    LAI_MEDIUM            = 2,
    LAI_HIGH               = 3,
    LAI_TOP                = 4
} ATTPriority_t;

typedef struct ATTUserEnteredCode_t
{
    ATTUserEnteredCodeType_t    type;
    ATTUserEnteredCodeIndicator_t indicator;
    char                        data[25];
    DeviceID_t                  collectVDN;
} ATTUserEnteredCode_t;

typedef enum ATTUserEnteredCodeType_t
{
    UE_NONE                = -1,
    UE_ANY                  = 0,
    UE_LOGIN_DIGITS        = 2,
    UE_CALL_PROMPTER       = 5,
    UE_DATA_BASE_PROVIDED  = 17,
    UE_TONE_DETECTOR       = 32
} ATTUserEnteredCodeType_t;

typedef enum ATTUserEnteredCodeIndicator_t
{
    UE_COLLECT = 0,
    UE_ENTERED = 1
} ATTUserEnteredCodeIndicator_t;

typedef struct ATTUserToUserInfo_t
{
    ATTUUIProtocolType_t    type;
    struct {
        short                length; // 0 indicates UUI not present
        unsigned char        value[33];
    } data;
} ATTUserToUserInfo_t;

typedef enum ATTUUIProtocolType_t
{
    UUI_NONE      = -1,    // indicates not specified
    UUI_USER_SPECIFIC= 0,  // user-specific
    UUI_IA5_ASCII = 4      // null terminated ascii char string
} ATTUUIProtocolType_t;

```

Route Request Service (TSAPI Version 1)

Direction: Switch to Client

Event: CSTARouteRequestEvent

Service Parameters: routeRegisterReqID, routingCrossRefID, currentRoute, callingDevice, routedCall, routedSelAlgorithm, priority, setupInformation

Functional Description:

The switch sends a CSTARouteRequestEvent to request a destination for a call arrived on a routing device from a routing server application. The application may have registered as the routing server for the routing device on the switch that is making the request, or it may have registered as the default routing server. The CSTARouteRequestEvent includes call-related information. A routing server application typically uses the call-related information and a database to determine the destination for the call. The routing server application responds to the CSTARouteRequestExtEvent via a cstaRouteSelect () request that specifies a destination for the call or a cstaRouteEnd () request, if the application has no destination for the call.

Detailed Information:

- The first cstaRouteSelect() response received by the switch is used as the route for the call, and all other CSTARouteRequestEvents for the call are canceled via CSTARouteEndEvents.
- If application terminates the CSTARouteRequestEvent request via a cstaRouteEnd(), the switch continues vector processing.
- A CSTARouteRequestEvent request will not affect the Call Event Reports.

Syntax

```

#include <acs.h>
#include <csta.h>

// CSTARouteRequestEvent - CSTA Unsolicited Event

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTAREQUEST
    EventType_t eventType; // CSTA_ROUTE_REQUEST
} ACSEventHeader_t;

typedef long          RouteRegisterReqID_t;

typedef long          RoutingCrossRefID_t;

typedef char          DeviceID_t[64];

typedef struct ExtendedDeviceID_t {
    DeviceID_t          deviceID;
    DeviceIDType_t      deviceIDType;
    DeviceIDStatus_t    deviceIDStatus;
} ExtendedDeviceID_t;

typedef ExtendedDeviceID_t CallingDeviceID_t;

typedef ExtendedDeviceID_t CalledDeviceID_t;

typedef enum ConnectionID_Device_t {
    STATIC_ID = 0,
    DYNAMIC_ID = 1
} ConnectionID_Device_t;

typedef struct ConnectionID_t {
    long                callID;
    DeviceID_t          deviceID;
    ConnectionID_Device_t devIDType;
} ConnectionID_t;

typedef enum SelectValue_t {
    SV_NORMAL = 0,
    SV_LEAST_COST = 1,
    SV_EMERGENCY = 2,
    SV_ACD = 3,
    SV_USER_DEFINED = 4
} SelectValue_t;

```

Syntax (Continued)

```
typedef struct SetupValues_t {
    int                length;
    unsigned char     *value;
} SetupValues_t;

typedef struct {
    RouteRegisterReqID_t    routeRegisterReqID;
    RoutingCrossRefID_t    routingCrossRefID;
    CalledDeviceID_t       currentRoute;
                           // TSAPI/cstadevs.h is wrong
    CallingDeviceID_t      callingDevice;
                           // TSAPI/cstadevs.h is wrong
    ConnectionID_t         routedCall;
    SelectValue_t          routedSelAlgorithm;
    Boolean                priority;
    SetupValues_t          setupInformation;
} CSTARouteRequestEvent_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID; // Unused for Route Request Event
            union
            {
                CSTARouteRequestEvent_t routeRequest;
            } u;
        } cstaRequest;
    } event;
    char    heap[CSTA_MAX_HEAP];
} CSTAEvent_t;
```

Route Select Service (TSAPI Version 2)

Direction: Client to Switch

Function: cstaRouteSelectInv()

**Private Data Function: attV6RouteSelect() (private data version 6),
attRouteSelect() (private data version 2, 3, 4, and 5)**

**Service Parameters: routeRegisterReqID, routingCrossRefID,
routeSelected, remainRetry, setupInformation, routeUsedReq**

**Private Parameters: callingDevice, directAgentCallSplit, priorityCalling,
destRoute, collectCode, userProvidedCode, userInfo**

Ack Parameters: noData

Nak Parameter: universalFailure

Functional Description:

The routing server application uses cstaRouteSelectInv() to provide a destination to the switch in response to a CSTARouteRequestExtEvent for a call.

Service Parameters:

<i>routeRegisterReqID</i>	[mandatory] Contains a handle to the routing registration session for which the application is providing routing service. The routing server application received this handle in a <code>CSTARouteRegisterReqConfEvent</code> confirmation to a <code>cstaRouteRegisterReq()</code> request.
<i>routingCrossRefID</i>	[mandatory] Contains the handle for the routing dialog of this call. The application previously received this handle in the <code>CSTARouteRequestExtEvent</code> for the call.
<i>routeSelected</i>	[mandatory] Specifies a destination for the call. If the destination is an off-PBX number, it can contain the TAC/ARS/AAR information (see <code>destRoute</code>).
<i>remainRetry</i>	[optional - not supported] Specifies the number of times that the application is willing to receive a <code>CSTARouteRequestExtEvent</code> for this call in case the switch needs to request an alternate route.
<i>setupInformation</i>	[optional - not supported] Contains a revised ISDN call setup message that the switch will use to route the call.
<i>routeUsedReq</i>	[optional - supported] Indicates a request to receive a <code>CSTARouteUsedExtEvent</code> for the call.

NOTE:

If specified, the G3PD always returns the same destination information that is specified in the `routeSelected` and `destRoute` of this `cstaRouteSelectInv()`.

Private Parameters:

<i>callingDevice</i>	[optional] Specifies the calling device. A NULL specifies that this parameter is not present.
<i>directAgentCallSplit</i>	[optional] Specifies the ACD agent's split extension for a Direct-Agent call routing. A Direct-Agent call is a special type of ACD call that is directed to a specific agent rather than to any available agent. The agent specified in the <code>routeSelected</code> must be logged into this split. A NULL parameter specifies that this is not a Direct-Agent call.
<i>priorityCalling</i>	[mandatory] Specifies the priority of the call. Values are "On" (TRUE) or "Off" (FALSE). When "On" is selected, a priority call is placed if the <code>routeSelected</code> is an on-PBX destination. When "On" is selected for an off-PBX <code>calledDevice</code> , the call will be denied.

<i>destRoute</i>	[optional] Specifies the TAC/ARS/AAR information for off-PBX destinations, if the information is not included in the routeSelected. A NULL parameter specifies no TAC/ARS/AAR information.
<i>collectCode</i>	[optional] This parameter allows the application to request that a DTMF tone detector (TN744) be connected to the routed call and to detect and collect caller (call originator) entered code/digits. <ul style="list-style-type: none">■ These digits are collected while the call is not in vector processing. The switch handles these digits like dial-ahead digits, and they may be used by Call Prompting features. The code/digits collected are passed to the application via event reports.■ The collectParty parameter in collectCode indicates to which party on the call the tone detector should listen. Currently, the call originator is the only option supported.■ The collectCode and userProvidedCode are mutually exclusive. If collectCode is present, then userProvidedCode cannot be present. A NULL indicates this parameter is not specified. If the collectCode type is set to "UC_NONE", it also indicates that no collectCode is sent with this request.
<i>userProvidedCode</i>	[optional] This parameter allows the application to send code/digits (ASCII string with 0-9, *, and # only) with the routed call. These code/digits are treated as dial-ahead digits for the call, and are stored in a dial-ahead digit buffer. <ul style="list-style-type: none">■ They can be collected (one at a time or in a group) using the collect digits vector command(s) on the switch.■ The userProvidedCode and collectCode parameters are mutually exclusive. If userProvidedCode is present, then collectCode cannot be present.■ A NULL indicates no user provided code. If the userProvidedCode type is set to "UP_NONE", it also indicates no userEnteredCode is sent with this request.■ The # character terminates the G3 PBX collection of user input so it is the last character present in the string if it is sent.¹

- Application designers must be aware that if a user enters more digits than requested, the excess digits remain in the G3 PBX prompting buffer and may therefore interfere with any later digit collection or reporting.

userInfo

[optional] Contains user-to-user information. This parameter allows the application to associate caller information, up to 32 or 96 bytes, with a call. It may be a customer number, credit card number, alphanumeric digits, or a binary string.

It is propagated with the call whether the call is routed to a destination on the local switch or to a destination on a remote switch over PRI trunks. The switch sends the user-to-user information (UUI) in the ISDN SETUP message over the PRI trunk to establish the call. The local and the remote switch include the UUI in the Delivered Event Report and in the CSTARouteRequestExtEvent to the application. A NULL indicates that this parameter is not present.

Prior to G3V8, the maximum length of userInfo was 32 bytes. Beginning with G3V8, the maximum length of userInfo was increased to 96 bytes.

⇒ NOTE:

An application using private data version 5 and earlier can only receive a maximum of 32-byte data for userInfo, regardless of the size data that is sent by the switch.

The following UUI protocol types are supported:

- UUI_NONE — There is no data provided in the data parameter.
- UUI_USER_SPECIFIC — The content of the data parameter is a binary string. The correct size (maximum of 32 or 96 bytes) of data must be specified in the size parameter.
- UUI_IA5_ASCII — The content of the data parameter must be a null-terminated IA5 (ASCII) character string. The correct size (maximum of 32 or 96 bytes excluding the null terminator) of data must be specified in the size parameter.

-
1. The user-to-user code collection stops when the user enters the requested number of digits or enters a # character to end the digit entry. If a user enters the # before entering the requested number of digits, then the # appears in the character string.

Ack Parameters:

noData None for this service.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain one of the following error values, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- INVALID_CSTA_DEVICE_IDENTIFIER (12) An invalid routeRegisterReqID has been specified in the Route Select request.
- INVALID_CROSS_REF_ID (17) An invalid routeCrossRefID has been specified in the Route Select request.

Detailed Information:

An application may receive one CSTARouteEndEvent and one universalFailure for a cstaRouteSelectInv() request for the same call in one of the following call scenarios:

- Switch/G3PD sends a CSTARouteRequestExtEvent to application.
- Caller drops the call.
- Application sends a CSTARouteSelectInv() request to G3PD.
- Switch/G3PD sends a CSTARouteEndEvent (errorValue = NO_ACTIVE_CALL) to application.
- G3PD receives the CSTARouteSelectInv() request, but call has been dropped.
- G3PD sends universalFailure for the CSTARouteSelectInv() request (errorValue = INVALID_CROSS_REF_ID) to application.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaRouteSelectInv() - Service Request

RetCode_t cstaRouteSelectInv(
    ACSHandle_t      acsHandle,
    InvokeID_t       invokeID, // V1 & V2 are different here
    RouteRegisterReqID_t routeRegisterReqID,
    RoutingCrossRefID_t routingCrossRefID,
    DeviceID_t       *routeSelected,
    RetryValue_t     remainRetry,
    SetUpValues_t    *setUpInformation,
    Boolean          routeUsedReq,
    PrivateData_t    *privateData);

typedef long        RouteRegisterReqID_t;

typedef long        RoutingCrossRefID_t;

typedef char        DeviceID_t[64];

typedef short       RetryValue_t;

typedef struct SetUpValues_t {
    int              length;
    unsigned char    *value;
} SetUpValues_t;
```

Private Data Version 6 Syntax

```

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attV6RouteSelect() - Service Request Private Data Setup Function

RetCode_t    attV6RouteSelect(
    ATTPrivateData_t*attPrivateData,           // length must be
                                                // set
    DeviceID_t    *callingDevice              // not in use
    DeviceID_t    *directAgentCallSplit,      // ACD Agents
                                                // split
    Boolean        priorityCalling,           // TRUE = On,
                                                // FALSE = Off
                                                // (or not
                                                // specified)
    DeviceID_t    *destRoute,                 // TAC/ARS/AAR for
                                                // off-PBX ext
    ATTUserCollectCode_t*collectCode,         // Request DTMF
                                                // tone detector
    ATTUserProvidedCode_t*userProvidedCode,   // Code to send
                                                // with routed
                                                // call
    ATTUserToUserInfo_t*userInfo);           // user-to-user
                                                // info with call

typedef struct ATTPrivateData_t {
    char                vendor[32];
    ushort              length;
    char                data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef struct ATTUserCollectCode_t {
    ATTCollectCodeType_ttype;
    short               digitsToBeCollected; // must be 1 - 24 digits
    short               timeout;             // must be 0 - 63 seconds
    ConnectionID_tcollectParty; // reserved - not in use
                                                // (defaults to call originator)
    ATTSpecificEvent_tspecificEvent; // Ignored (Defaults to Far
                                                // End Answer)
} ATTUserCollectCode_t;

typedef enum ATTCollectCodeType_t {
    UC_NONE                = 0, // indicates UCC not present
    UC_TONE_DETECTOR      = 32
} ATTCollectCodeType_t;

typedef enum ATTSpecificEvent_t {
    SE_ANSWER                = 11,
    SE_DISCONNECT = 4
} ATTSpecificEvent_t;

```

Private Data Version 6 Syntax (Continued)

```
typedef struct ATTUserProvidedCode_t {
ATTProvidedCodeType_t  type;
    char                data[25];
} ATTUserProvidedCode_t;

typedef enum ATTProvidedCodeType_t {
    UP_NONE = 0,          // indicates UPC not present
    UP_DATA_BASE_PROVIDED = 17
} ATTProvidedCodeType_t;

#define ATT_MAX_USER_INFO 129
#define ATT_MAX_UII_SIZE 96
#define ATTV5_MAX_UII_SIZE 32

typedef struct ATTUserToUserInfo_t {
    ATTUIIProtocolType_t type;
    struct {
        short          length; // 0 indicates UII not present
        unsigned char  value[ATT_MAX_USER_INFO];
    } data;
} ATTUserToUserInfo_t;

typedef enum ATTUIIProtocolType_t {
    UII_NONE= -1          // indicates not specified
    UII_USER_SPECIFIC= 0, // user-specific
    UII_IA5_ASCII= 4      // null terminated ascii
                          // character string
} ATTUIIProtocolType_t
```

Private Data Version 2-5 Syntax

```

#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attRouteSelect() - Service Request Private Data Setup Function

RetCode_t    attRouteSelect(
    ATTPrivateData_t*attPrivateData,           // length must be
                                                // set
    DeviceID_t  *callingDevice                 // not in use
    DeviceID_t  *directAgentCallSplit,        // ACD Agents
                                                // split
    Boolean     priorityCalling,              // TRUE = On,
                                                // FALSE = Off
                                                // (or not
                                                // specified)
    DeviceID_t  *destRoute,                   // TAC/ARS/AAR for
                                                // off-PBX ext
    ATTUserCollectCode_t*collectCode,         // Request DTMF
                                                // tone detector
    ATTUserProvidedCode_t*userProvidedCode,   // Code to send
                                                // with routed
                                                // call
    ATTUserToUserInfo_t*userInfo);           // user-to-user
                                                // info with call

typedef struct ATTPrivateData_t {
    char                vendor[32];
    ushort              length;
    char                data[ATT_MAX_PRIVATE_DATA];
} ATTPrivateData_t;

typedef struct ATTUserCollectCode_t {
    ATTCollectCodeType_ttype;
    short               digitsToBeCollected; // must be 1 - 24 digits
    short               timeout;             // must be 0 - 63 seconds
    ConnectionID_tcollectParty; // reserved - not in use
                                                // (defaults to call originator)
    ATTSpecificEvent_tspecificEvent; // Ignored (Defaults to Far
                                                // End Answer)
} ATTUserCollectCode_t;

typedef enum ATTCollectCodeType_t {
    UC_NONE              = 0, // indicates UCC not present
    UC_TONE_DETECTOR    = 32
} ATTCollectCodeType_t;

typedef enum ATTSpecificEvent_t {
    SE_ANSWER            = 11,
    SE_DISCONNECT = 4
} ATTSpecificEvent_t;

```

Private Data Version 2-5 Syntax (Continued)

```
typedef struct ATTUserProvidedCode_t {
ATTProvidedCodeType_t  type;
    char                data[25];
} ATTUserProvidedCode_t;

typedef enum ATTProvidedCodeType_t {
    UP_NONE = 0,          // indicates UPC not present
    UP_DATA_BASE_PROVIDED = 17
} ATTProvidedCodeType_t;

typedef struct ATTUserToUserInfo_t {
    ATTUUIProtocolType_t type;
    struct {
        short          length; // 0 indicates UUI not present
        unsigned char  value[32];
    } data;
} ATTUserToUserInfo_t;

typedef enum ATTUUIProtocolType_t {
    UUI_NONE= -1          // indicates not specified
    UUI_USER_SPECIFIC= 0, // user-specific
    UUI_IA5_ASCII= 4      // null terminated ascii
                          // character string
} ATTUUIProtocolType_t
```

Route Select Service (TSAPI Version 1)

Direction: Client to Switch

Function: cstaRouteSelect()

Service Parameters: routeRegisterReqID, routingCrossRefID, routeSelected, remainRetry, setupInformation, routeUsedReq

Functional Description:

The routing server application uses cstaRouteSelect () to provide a destination to the switch in response to a CSTARouteRequestEvent for a call.

Detailed Information:

An application may receive two CSTARouteEndEvent(s) for a cstaRouteSelect() request for the same call in one of the following call scenarios:

- Switch/G3PD sends a CSTARouteRequestEvent to application.
- Caller drops the call.
- Application sends a CSTARouteSelect() request to G3PD.
- Switch/G3PD sends a CSTARouteEndEvent (errorValue = NO_ACTIVE_CALL) to application.
- G3PD receives the CSTARouteSelect() request, but call has been dropped.
- G3PD sends a CSTARouteEndEvent for the CSTARouteSelect() request (errorValue = INVALID_CROSS_REF_ID) to application.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaRouteSelect() - Service Request

RetCode_t    cstaRouteSelect (
    ACSHandle_t    acsHandle,
    RouteRegisterReqID_t routeRegisterReqID,
    RoutingCrossRefID_t routingCrossRefID,
    DeviceID_t    *routeSelected,
    RetryValue_t    remainRetry,
    SetUpValues_t    *setupInformation,
    Boolean        routeUsedReq,
    PrivateData_t    *privateData);

typedef long    RouteRegisterReqID_t;

typedef long    RoutingCrossRefID_t;

typedef char    DeviceID_t[64];

typedef short    RetryValue_t;

typedef struct SetUpValues_t {
    int    length;
    unsigned char*value;
} SetUpValues_t;
```


Route Used Event (TSAPI Version 2)

Direction: Switch to Client

Event: CSTARouteUsedExtEvent

Private Data Event: ATTRouteUsedEvent

Service Parameters: routeRegisterReqID, routingCrossRefID, routeUsed, callingDevice, domain

Private Parameters: destRoute

Functional Description:

The switch uses a CSTARouteUsedExtEvent to provide a destination to the routing server application with the actual destination of a call for which the application previously sent a V containing a destination. The routeUsed and destRoute parameters contain the same information specified in the routeSelected and destRoute parameters of the previous cstaRouteSelectInv() request of this call, respectively. The callingDevice parameter contains the same calling device number provided in the previous CSTARouteRequestExtEvent of this call.

Service Parameters:

<i>routeRegisterReqID</i>	[mandatory] Contains a handle to the routing registration session for which the application is providing routing service. The routing server application received this handle in a <code>CSTARouteRegisterReqConfEvent</code> confirmation to a <code>cstaRouteRegisterReq()</code> request.
<i>routingCrossRefID</i>	[mandatory] Contains the handle for the routing dialog of this call. The application previously received this handle in the <code>CSTARouteRequestExtEvent</code> for the call.
<i>routeUsed</i>	[mandatory] Specifies the destination of the call. This parameter has the same destination specified in the <code>routeSelected</code> of the previous <code>cstaRouteSelectInv()</code> request of this call.
<i>callingDevice</i>	[optional - supported] Specifies the call origination device. It contains the same calling device number provided in the previous <code>CSTARouteRequestExtEvent</code> .
<i>domain</i>	[optional - not supported] Indicates whether the call has left the switching domain accessible to the G3PD. Typically, a call leaves a switching domain when it is routed to a trunk connected to another switch or to the public switch network. This parameter is not supported and is always set to <code>FALSE</code> . This does not mean that the call has (or has not) left the G3 switch. An application should ignore this parameter

Private Parameters:

<i>destRoute</i>	[optional] Specifies the TAC/ARS/AAR information for off-PBX destinations. This parameter contains the same information specified in the <code>destRoute</code> of the previous <code>cstaRouteSelectInv()</code> request of this call.
-------------------------	---

Detailed Information:

- Note that the number provided in the `routeUsed` parameter is from the `routeSelected` parameter of the previous `cstaRouteSelectInv()` request of this call received by the G3PD. This information in `routeUsed` is not from the G3 PBX and it may not represent the true route that the G3 PBX used.
- Note that the number provided in the `destRoute` parameter is from the `destRoute` parameter of the previous `cstaRouteSelectInv()` request of this call received by the G3PD. This information in `destRoute` is not from the G3 PBX and it may not represent the true route that the G3 PBX used.
- The number provided in the `callingDevice` parameter is from the `callingDevice` parameter of the previous `CSTARouteRequestExtEvent` of this call sent by the G3PD.

Syntax

```

#include <acs.h>
#include <csta.h>

// CSTARouteUsedExtEvent - Route Select Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTAEVENTREPORT
    EventType_t eventType; // CSTA_ROUTE_USED_EXT
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;

    union
    {
        struct
        {
            union
            {
                CSTARouteUsedExtEvent_t routeUsed;
            } u;
        } cstaEventReport;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTARouteUsedExtEvent_t {
    RouteRegisterReqID_t routeRegisterReqID;
    RoutingCrossRefID_t routingCrossRefID;
    DeviceID_t routeUsed; // V1 & V2 are different here
    DeviceID_t callingDevice; // TSAPI V1 & V2 are different here
    Boolean domain;
} CSTARouteUsedExtEvent_t;

```

Route Used Event (TSAPI Version 1)

Direction: Switch to Client

Event: CSTARouteUsedEvent

Service Parameters: routeRegisterReqID, routingCrossRefID, routeUsed, callingDevice, domain

Functional Description:

The switch uses a CSTARouteUsedExtEvent to provide a destination to the routing server application with the actual destination of a call for which the application previously sent a V containing a destination. The routeUsed and destRoute parameters contain the same information specified in the routeSelected and destRoute parameters of the previous cstaRouteSelectInv() request of this call, respectively. The callingDevice parameter contains the same calling device number provided in the previous CSTARouteRequestExtEvent of this call.

Detailed Information:

- The number provided in the routeUsed parameter is from the routeSelected parameter of the previous cstaRouteSelect() request of this call received by the G3PD.
- The number provided in the callingDevice parameter is from the callingDevice parameter of the previous CSTARouteRequestEvent of this call sent by the G3PD.

Syntax

```

#include <acs.h>
#include <csta.h>

// CSTARouteUsedEvent - Route Select Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTAEVENTREPORT
    EventType_t eventType; // CSTA_ROUTE_USED
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;

    union
    {
        struct
        {
            union
            {
                CSTARouteUsedEvent_t routeUsed;
            } u;
        } cstaEventReport;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTARouteUsedEvent_t {
    RouteRegisterReqID_t routeRegisterReqID;
    RoutingCrossRefID_t routingCrossRefID;
    DeviceID_t routeUsed;
    DeviceID_t callingDevice;
    Boolean domain;
} CSTARouteUsedEvent_t;

```

Private Parameter Syntax

If private data accompanies a `CSTARouteUsedExtEvent`, then the private data would be stored in the location that the application specified as the `privateData` parameter in the `acsGetEventBlock()` or `acsGetEventPoll()` request. If the `privateData` pointer is set to `NULL` in these requests, then `CSTARouteUsedExtEvent` does not deliver private data to the application.

If the `acsGetEventBlock()` or `acsGetEventPoll()` returns Private Data length of 0, then no private data is provided with this Route Request.

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTRouteUsedEvent - Service Response Private Data
typedef struct
{
    ATTEventTypeeventType; // ATT_ROUTE_USED
    union
    {
        ATTRouteUsedEvent_tdestRoute;
    }u;
} ATTEvent_t;

typedef struct ATTRouteUsedEvent_t
{
    DeviceID_t destRoute;
} ATTRouteUsedEvent_t;
```

Overview

The System Status Services allow an application to receive reports on the status of the switching system. (System Status services with the driver/switch as the client are not supported.)

The following System Status Services and Events are available:

System Status Request Service — `cstaSysStatReq()`

This service is used by a client application to request system status information from the driver/switch domain.

System Status Start Service — `cstaSysStatStart()`

This service allows an application to register for System Status event reporting.

System Status Stop Service — `cstaSysStatStop()`

This service allows an application to cancel a previously registered request for System Status event reporting.

Change System Status Filter Service — cstaChangeSysStatFilter()

This service allows an application to request a change in the filter options for System Status event reporting.

System Status Event — CSTASysStatEvent

This unsolicited event informs the application of changes in the system status of the driver/switch.

System Status Events — Not Supported

The following System Status Events are not supported:

- System Status Request Event – CSTASysStatReqEvent
- System Status Request Confirmation – cstaSysStatReqConf()
- System Status Event Send – cstaSysStatEventSend()

System Status Request Service

Direction: Client to Switch

Function: cstaSysStatReq()

Confirmation Event: CSTASysStatReqConfEvent

Service Parameters: none

Ack Parameters: systemStatus

Ack Private Parameters: count, plinkStatus (private data version 5),
linkStatus (private data versions 2, 3, and 4)

Nak Parameter: universalFailure

Functional Description:

This service is used by a client application to request system status information from the driver/switch.

Service Parameters:

noData None for this service.

Ack Parameters:

systemStatus [mandatory — partially supported] Provides the application with a cause code defining the overall system status as follows:

- **NORMAL** — This status indication indicates that at least one CTI link to the switch is available. The system status is normal, and TSAPI requests and responses are enabled.
- **DISABLED** — This system status indicates that there is no available CTI link to the switch. The **DISABLED** status implies that there are no active Monitor requests or Route Register sessions. TSAPI requests and responses are disabled and reject responses should be provided for each request or response.

Ack Private Parameters:

count Identifies the number of CTI links described in the *plinkStatus* private ack parameter.

plinkStatus Specifies the status of each CTI link to the switch. The G3PD driver supports multiple CTI links between the Telephony Server and the switch for enhanced throughput and redundancy. The routing of TSAPI service requests and responses over the individual CTI links by the G3PD is hidden from the application.

(The TSAPI application programmer does not need to consider the individual CTI links to a switch when sending/receiving TSAPI service requests/responses.) The *plinkStatus* private data parameter may be used to check the availability of each administered CTI link to which the G3 switch the application is connected. The status of each link identified by *linkID* will be set to one of the following values in the *linkState* field:

- **LS_LINK_UP** — The link is able to support telephony services to the switch.
- **LS_LINK_DOWN** — The link is unable to support telephony services to the switch.

- LS_LINK_UNAVAIL — The link has been disabled (busied-out) via the OA&M interface and will not support new CSTA requests. Existing telephony service requests maintained by this link will continue.

This parameter is supported by private data version 5 and later only.

linkStatus

Specifies the status of each CTI link to the switch. For details, see the description for the plinkStatus private ack parameter. This parameter is supported by private data versions 2, 3, and 4.

Nak Parameter:

universalFailure

If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

Detailed Information:

- Multiple CTI Links — If multiple CTI links are connected and administered to a specific switch, the systemStatus parameter will indicate the aggregate link status. If at least one CTI link is available to support TSAPI requests and responses, the systemStatus will be set to NORMAL. If there are no CTI links to a switch able to support TSAPI requests and responses, the systemStatus will be set to DISABLED.
- If multiple CTI links are connected and administered to a specific switch, Private Data must be used to determine if the switching system is performing as administered. The plinkStatus private parameter can be used to check the status of each individual CTI link.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaSysStatReq() - Request for system status

RetCode_t cstaSysStatReq (
    ACSHandle_t          acsHandle,
    InvokeID_t          invokeID,
    PrivateData_t FAR   *privateData);

// CSTASysStatReqConfEvent - System status confirmation event

typedef struct
{
    ACSHandle_t  acsHandle;
    EventClass_t eventClass; // CSTACONFIRMATION
    EventType_t  eventType;  // CSTA_SYS_STAT_REQ_CONF
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            union
            {
                CSTASysStatReqConfEvent_t sysStatReq;
            } u;
            cstaEventReport;
        } event;
        char heap[CSTA_MAX_HEAP];
    }
} CSTAEvent_t;

typedef struct CSTASysStatReqConfEvent_t {
    SystemStatus_t  systemStatus;
} CSTASysStatReqConfEvent_t;

typedef enum SystemStatus_t {
    SS_INITIALIZING      = 0, // Not supported
    SS_ENABLED           = 1, // Not supported
    SS_NORMAL            = 2, // Supported
    SS_MESSAGES_LOST    = 3, // Not supported
    SS_DISABLED         = 4, // Supported
    SS_OVERLOAD_IMMINENT = 5, // Not supported
    SS_OVERLOAD_REACHED = 6, // Not supported
    SS_OVERLOAD_RELIEVED = 7, // Not supported
} SystemStatus_t;
```

Private Data Version 5 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTLinkStatusEvent - Service Response Private Data

typedef struct
{
    ATTEventType_t eventType; // ATT_LINK_STATUS
    union
    {
        ATTLinkStatusEvent_t linkStatus;
    } u;
} ATTEvent_t;

typedef struct ATTLinkStatusEvent_t
{
    int count;
    ATTLinkStatus_t FAR *pLinkStatus;
} ATTLinkStatusEvent_t;

typedef struct ATTLinkStatus_t
{
    short linkID;
    ATTLinkState_t linkState;
} ATTLinkStatus_t;

typedef enum ATTLinkState_t {
    LS_LINK_UNAVAIL= 0, // the link is disabled
    LS_LINK_UP = 1, // the link is up
    LS_LINK_DOWN= 2 // the link is down
} ATTLinkState_t;
```

Private Data Version 4 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV4LinkStatusEvent - Service Response Private Data

typedef struct
{
    ATTEventType_t eventType; // ATTV4_LINK_STATUS
    union
    {
        ATTV4LinkStatusEvent_t tv4linkStatus;
    } u;
} ATTEvent_t;

typedef struct ATTV4LinkStatusEvent_t
{
    short count;
    ATTLinkStatus_t linkStatus[8];
} ATTV4LinkStatusEvent_t;

typedef struct ATTLinkStatus_t
{
    short linkID;
    ATTLinkState_t linkState;
} ATTLinkStatus_t;

typedef enum ATTLinkState_t {
    LS_LINK_UNAVAIL= 0, // the link is disabled
    LS_LINK_UP = 1, // the link is up
    LS_LINK_DOWN= 2 // the link is down
} ATTLinkState_t;
```

Private Data Versions 2 and 3 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV3LinkStatusEvent - Service Response Private Data

typedef struct
{
    ATTEventType_t eventType; // ATTV3_LINK_STATUS
    union
    {
        ATTV3LinkStatusEvent_t tv3linkStatus;
    } u;
} ATTEvent_t;

typedef struct ATTV3LinkStatusEvent_t
{
    short count;
    ATTLINKSTATUS_t linkStatus[4];
} ATTV3LinkStatusEvent_t;

typedef struct ATTLINKSTATUS_t
{
    short linkID;
    ATTLINKSTATE_t linkState;
} ATTLINKSTATUS_t;

typedef enum ATTLINKSTATE_t {
    LS_LINK_UNAVAIL= 0, // the link is disabled
    LS_LINK_UP = 1, // the link is up
    LS_LINK_DOWN= 2 // the link is down
} ATTLINKSTATE_t;
```

System Status Start Service

Direction: Client to Switch

Function: cstaSysStatStart()

Confirmation Event: CSTASysStatStartConfEvent

Private Data Function: attSysStat()

Service Parameters: statusFilter

Private Parameters: linkStatReq

Ack Parameters: statusFilter

Ack Private Parameters: count, plinkStatus (private data version 5),

linkStatus (private data versions 2, 3, and 4)

Nak Parameter: universalFailure

Functional Description:

This service allows the application to register for System Status event reporting from the driver/switch. The application can register to receive a CSTASysStatEvent each time the status of the driver/switch changes. The service request includes a filter so the application can filter those status events that are not of interest to the application. Only one cstaSysStatStart() request is allowed for an acsOpenStream() request. If one exists, the second one will be rejected.

Service Parameters:

statusFilter [mandatory — partially supported] A filter used to specify the system status events that are not of interest to the application. If a bit in *statusFilter* is set to TRUE (1), the corresponding event will not be sent to the application. The only System Status events that will be reported are SS_ENABLED, SS_NORMAL and SS_DISABLED. A request to filter any other System Status events will be ignored.

Private Parameters:

linkStatReq [optional] The application can use the *linkStatReq* private parameter to request System Status events for changes in the state of individual G3 switch CTI links. The *linkStatReq* private parameter is only useful for multilink configurations.

- If *linkStatReq* is set to TRUE (ON), System Status Event Reports will be sent for changes in the states of each individual CTI link. When a CTI link changes between up (LS_LINK_UP), down (LS_LINK_DOWN), or unavailable/busied-out (LS_LINK_UNAVAIL), a System Status Event Report will be sent to the application. The private data in the System Status Event Report will include the link ID and state for each CTI link to the G3 switch, and not just the link ID and state of the CTI link that experienced a state transition.
- If the *linkStatReq* private parameter was not specified or set to FALSE, changes in the states of individual G3 CTI links will not result in System Status Event Reports unless all links are down, or the first link is established. (The System Status Event Report is always sent when all links are down, or when the first link is established from an “all CTI links down” state.)

Ack Parameters:

statusFilter [optional — partially supported] Specifies the System Status Event Reports that are to be filtered before they reach the application. The *statusFilter* may not be the same as the *statusFilter* specified in the service request, because filters for System Status Events that are not supported are always turned on (TRUE) in *systemFilter*.

The following filters will always be set to ON, meaning that there are no reports supported for these events:

- SF_INITIALIZING
- SF_MESSAGES_LOST

- SF_OVERLOAD_IMMINENT
- SF_OVERLOAD_REACHED
- SF_OVERLOAD_RELIEVED

Ack Private Parameters:

count Identifies the number of CTI links described in the plinkStatus private ack parameter. This parameter is only provided when the linkStatusReq private parameter was set to TRUE.

plinkStatus Specifies the status of each CTI link to the switch. This parameter is only provided when the linkStatusReq private parameter was set to TRUE. The plinkStatus private data parameter will indicate the availability of each administered CTI link to the G3 switch to which the application is connected.

The status of each link identified by linkID will be set to one of the following values in the linkState field:

- LS_LINK_UP — The link is able to support telephony services to the switch.
- LS_LINK_DOWN — The link is unable to support telephony services to the switch.
- LS_LINK_UNAVAIL —The link has been disabled (busied-out) via the OA&M interface and will not support new CSTA requests. Existing telephony service requests maintained by this link will continue.

This parameter is supported by private data version 5 and later only.

linkStatus Specifies the status of each CTI link to the switch. For details, see the description for the plinkStatus private ack parameter. This parameter is supported by private data versions 2, 3, and 4.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error value, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- GENERIC_OPERATION_REJECTION (71) Only one cstaSysStatStart() request is allowed for an acsOpenStream() request. If one exists, the second one will be rejected.

Issue 1 — December 2001

Detailed Information:

- The linkStatReq private parameter is only useful in multilink configurations.
- Only one cstaSysStatStart() request is allowed for an acsOpenStream() request. If one exists, the second one will be rejected. An application can cancel a request for System Status event reporting via cstaSysStatusStop(), and then issue a subsequent cstaSysStatStart() request.
- If the application requests System Status Event Reports for changes in specific CTI link states (up/down/unavailable), it must examine the private data included in the CSTASysStatEvent Event Report to determine the changes in the individual CTI links.
- The count and plinkStatus private ack parameters will only be provided when the linkStatReq parameter was set to TRUE in the System Status Start service request.
- A CSTASysStatEvent event report will be sent with the systemStatus set to SS_DISABLED when the last CTI link to the G3 switch has failed. The application can examine the private data portion of the event report, but it will always indicate that all CTI links are down (LS_LINK_DOWN) or unavailable (LS_LINK_UNAVAILABLE). All Call and Device Monitors will be terminated, all Routing Sessions will be aborted, and all outstanding CSTA requests should be negatively acknowledged.
- A CSTASysStatEvent Event Report will be sent with the systemStatus set to SS_ENABLED when the first CTI link to the G3 switch has been established from an "all CTI links down" state. The application can examine the private data portion of the Event Report to determine which CTI links are up (LS_LINK_UP), which CTI links are down (LS_LINK_DOWN), and which CTI links are disabled via the Telephony Services Administrator interface (LS_LINK_UNAVAIL). No Call or Device Monitors, or Routing Sessions should exist at this point.
- A CSTASysStatEvent Event Report will be sent with the systemStatus set to SS_NORMAL when the application has requested event reports for changes in specific CTI link states (via the linkStatusReq private parameter) and a CTI link changes state to up, (LS_LINK_UP) down (LS_LINK_DOWN), or unavailable/busied-out via OA&M(LS_LINK_UNAVAIL).
Note that the systemStatus is set to SS_NORMAL, indicating that at least one CTI link to the switch is available. The application can examine the private data portion of the event report to determine which CTI links are up, down, or unavailable/busied-out. Call or Device Monitors, and Routing Sessions may have been terminated when the CTI link state changed to down (LS_LINK_DOWN).

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaSysStatStart() - Service Request

RetCode_t  cstaSysStatStart(
    ACSHandle_t      acsHandle,
    InvokeID_t       invokeID,
    SystemStatusFilter_t  statusFilter,
    PrivateData_tFAR *privateData);

typedef unsignedSystemStatusFilter_t;

#define SF_INITIALIZING      0x80    // Not supported
#define SF_ENABLED          0x40    // Supported
#define SF_NORMAL           0x20    // Supported
#define SF_MESSAGES_LOST    0x10    // Not supported
#define SF_DISABLED         0x08    // Supported
#define SF_OVERLOAD_IMMINENT 0x04    // Not supported
#define SF_OVERLOAD_REACHED 0x02    // Not supported
#define SF_OVERLOAD_RELIEVED 0x01    // Not supported

// CSTASysStatStartConfEvent - Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTACONFIRMATION
    EventType_t eventType;  // CSTA_SYS_STAT_START_CONF
} ACSEventHeader_t;

typedef struct CSTASysStatStartConfEvent_t {
    SystemStatusFilter_t  statusFilter;
} CSTASysStatStartConfEvent_t;
```

Syntax (Continued)

```
typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            InvokeID_t invokeID;
            union
            {
                CSTASysStatStartConfEvent_t
                sysStatStart;
            } u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;
```

Private Data Version 5 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attSysStat() - Service Request Private Data Setup Function

RetCode_t    attSysStat(
    ATTPrivateData_tFAR *attPrivateData, // length must be set
    Boolean      linkStatusReq); // send event reports for CTI
                                // link state changes

typedef struct ATTPrivateData_t
{
    char          vendor[32];
    unsigned short length;
    char          data[ATT_MAX_PRIVATE_DATA];
}

// ATTLinkStatusEvent - Service Response Private Data

typedef struct
{
    ATTEventTypeeventType; // ATT_LINK_STATUS
    union
    {
        ATTLinkStatusEvent_tlinkStatus;
    } u;
} ATTEvent_t;

typedef struct ATTLinkStatusEvent_t
{
    short          count;
    ATTLinkStatus_tFAR *pLinkStatus;
} ATTLinkStatusEvent_t;

typedef struct ATTLinkStatus_t
{
    short          linkID;
    ATTLinkState_t linkState;
} ATTLinkStatus_t;

typedef enum ATTLinkState_t {
    LS_LINK_UNAVAIL= 0, // the link is disabled
    LS_LINK_UP    = 1, // the link is up
    LS_LINK_DOWN= 2    // the link is down
} ATTLinkState_t;
```

Private Data Version 4 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attSysStat() - Service Request Private Data Setup Function

RetCode_t    attSysStat(
    ATTPrivateData_t *attPrivateData, // length must be set
    Boolean      linkStatusReq); // send event reports for CTI
                                // link state changes

typedef struct ATTPrivateData_t
{
    char          vendor[32];
    unsigned short length;
    char          data[ATT_MAX_PRIVATE_DATA];
}

// ATTV4LinkStatusEvent - Service Response Private Data

typedef struct
{
    ATTEventTypeeventType; // ATTV4_LINK_STATUS
    union
    {
        ATTV4LinkStatusEvent_tv4linkStatus;
    } u;
} ATTEvent_t;

typedef struct ATTV4LinkStatusEvent_t
{
    short          count;
    ATTLinkStatus_t linkStatus[8];
} ATTV4LinkStatusEvent_t;

typedef struct ATTLinkStatus_t
{
    short          linkID;
    ATTLinkState_t linkState;
} ATTLinkStatus_t;

typedef enum ATTLinkState_t {
    LS_LINK_UNAVAIL= 0, // the link is disabled
    LS_LINK_UP    = 1, // the link is up
    LS_LINK_DOWN= 2   // the link is down
} ATTLinkState_t;
```

Private Data Versions 2 and 3 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attSysStat() - Service Request Private Data Setup Function

RetCode_t    attSysStat(
    ATTPrivateData_t *attPrivateData, // length must be set
    Boolean      linkStatusReq); // send event reports for CTI
                                // link state changes

typedef struct ATTPrivateData_t
{
    char          vendor[32];
    unsigned short length;
    char          data[ATT_MAX_PRIVATE_DATA];
}

// ATTV3LinkStatusEvent - Service Response Private Data

typedef struct
{
    ATTEventType eventType; // ATTV3_LINK_STATUS
    union
    {
        ATTV3LinkStatusEvent_tv3linkStatus;
    } u;
} ATTEvent_t;

typedef struct ATTV3LinkStatusEvent_t
{
    short          count;
    ATTLinkStatus_t linkStatus[4];
} ATTV3LinkStatusEvent_t;

typedef struct ATTLinkStatus_t
{
    short          linkID;
    ATTLinkState_t linkState;
} ATTLinkStatus_t;

typedef enum ATTLinkState_t {
    LS_LINK_UNAVAIL= 0, // the link is disabled
    LS_LINK_UP    = 1, // the link is up
    LS_LINK_DOWN  = 2  // the link is down
} ATTLinkState_t;
```


System Status Stop Service

Direction: Client to Switch
Function: `cstaSysStatStop()`
Confirmation Event: `CSTASysStatStopConfEvent`
Service Parameters: none
Ack Parameters: none
Nak Parameter: `universalFailure`

Functional Description:

This service allows the application to cancel a previously registered monitor for System Status event reporting from the driver/switch domain

Service Parameters:

noData None for this service.

Ack Parameters:

noData None for this service.

Nak Parameter:

universalFailure If the request is not successful, the application will receive a `CSTAUniversalFailureConfEvent`. The error parameter in this event may contain one of the error values described in the “`CSTAUniversalFailureConfEvent`” section in Chapter 3:

Detailed Information:

- An application may receive `CSTASysStatEvents` from the driver/switch until the `CSTASysStatStopConfEvent` response is received. The application should check the confirmation event to verify that the System Status monitor has been deactivated.

After the G3 PBX driver has issued the `CSTASysStatStopConfEvent`, automatic notification of System Status Events will be terminated.

Syntax

```
#include <acs.h>
#include <csta.h>

// cstaSysStatStop() - Service Request

RetCode_t  cstaSysStatStop (
            ACSHandle_t      acsHandle,
            InvokeID_t       invokeID,
            PrivateData_t     *privateData);

// CSTASysStatStopConfEvent - Service Response

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTACONFIRMATION
    EventType_t eventType; // CSTA_SYS_STAT_STOP_CONF
} ACSEventHeader_t;

typedef char Nulltype;

typedef struct CSTASysStatStopConfEvent_t {
    Nulltype      null;
} CSTASysStatStopConfEvent_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            InvokeID_t invokeID;
            union
            {
                CSTASysStatStopConfEvent_t sysStatStop;
            } u;
        } cstaConfirmation;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;
```

Change System Status Filter Service

Direction: Client to Switch

Function: `cstaChangeSysStatFilter()`

Confirmation Event: `CSTAChangeSysStatFilterConfEvent`

Private Data Function: `attSysStat()`

Service Parameters: `statusFilter`

Private Parameters: `linkStatReq`

Ack Parameters: `statusFilterSelected`, `statusFilterActive`

Ack Private Parameters: `count`, `plinkStatus` (private data version 5),

`linkStatus` (private data versions 2, 3, and 4)

Nak Parameter: `universalFailure`

Functional Description:

This service allows the application to modify the filter used for System Status event reporting from the driver/switch domain. The application can filter those System Status events that it does not wish to receive. A `CSTASysStatEvent` will be sent to the application if the event occurs and the application has not specified a filter for that System Status Event. The application must have previously requested System Status Event reports via the `cstaSysStatStart()` request, else the `cstaChangeSysStatFilter()` will be rejected.

Service Parameters:

statusFilter [mandatory — partially supported] A filter used to specify the System Status Events that are not of interest to the application. If a bit in `statusFilter` is set to TRUE (1), the corresponding event will not be sent to the application. The only System Status Events that will be reported are `SS_ENABLED`, `SS_NORMAL` and `SS_DISABLED`. A request to filter any other System Status Events will be ignored.

Private Parameters:

linkStatReq [optional] The application can use the `linkStatReq` private parameter to request System Status Events for changes in the state of individual G3 switch CTI links.

- If `linkStatReq` is set to TRUE (ON), System Status Event Reports will be sent for changes in the states of each individual CTI link. When a CTI link changes between up (`LS_LINK_UP`), down (`LS_LINK_DOWN`), or unavailable/busied-out (`LS_LINK_UNAVAIL`), a System Status Event Report will be sent to the application. The private data in the System Status Event Report will include the link ID and state for each CTI link to the G3 switch, and not just the link ID and state of the CTI link that experienced a state transition.
- If the `linkStatReq` private parameter was set to FALSE, changes in the states of individual G3 CTI links will not result in System Status Event Reports unless all links are down, or the first link is established. (The System Status Event Report is always sent when all links are down, or when the first link is established from an “all links down” state.)
- If the `linkStatReq` private parameter was not specified, there will be no change in the reporting changes in the state of individual G3 CTI links. (If System Status Event Reports were sent for changes in individual G3 CTI links before a `cstaChangeStatFilter()` service request with no private data, the System Status Event Reports will continue to be sent after the `CSTAChangeSysStatFilterConfEvent` service response is received, and vice-versa.)

Ack Parameters:

statusFilterSelected [mandatory — partially supported] specifies the System Status Event Reports that are to be filtered before they reach the application. The *statusFilterSelected* may not be the same as the *statusFilter* specified in the service request, because filters for System Status Events that are not supported are always turned on in *statusFilterSelected*. The following filters will always be set to ON, meaning that there are no reports supported for these events:

- SF_INITIALIZING
- SF_MESSAGES_LOST
- SF_OVERLOAD_IMMINENT
- SF_OVERLOAD_REACHED
- SF_OVERLOAD_RELIEVED

statusFilterActive [mandatory — partially supported] Specifies the System Status Event Reports that were already active before the CSTAChangeSysStatConfEvent was issued by the driver. The following filters will always be set to ON, meaning that there are no reports supported for these events:

- SF_INITIALIZING
- SF_MESSAGES_LOST
- SF_OVERLOAD_IMMINENT
- SF_OVERLOAD_REACHED
- SF_OVERLOAD_RELIEVED

Ack Private Parameters:

count Identifies the number of CTI links described in the *plinkStatus* private ack parameter. This parameter is only provided when the *linkStatusReq* private parameter was set to TRUE.

plinkStatus Specifies the status of each CTI link to the switch. This parameter is only provided when the *linkStatusReq* private parameter was set to TRUE. The *plinkStatus* private data parameter will indicate the availability of each administered CTI link to the G3 to which the application is connected. The status of each link identified by *linkID* will be set to one of the following values in the *linkState* field:

- LS_LINK_UP — The link is able to support traffic.
- LS_LINK_DOWN — The link is unable to support traffic.

Issue 1 — December 2001

- LS_LINK_UNAVAIL — The link has been disabled (busied-out) via the OA&M interface and will not support new CSTA requests. Existing telephony service requests maintained by this link will continue.

This parameter is supported by private data version 5 and later only.

linkStatus

Specifies the status of each CTI link to the switch. For details, see the description for the plinkStatus private ack parameter. This parameter is supported by private data versions 2, 3, and 4.

Nak Parameter:

universalFailure

If the request is not successful, the application will receive a CSTAUniversalFailureConfEvent. The error parameter in this event may contain the following error value, or one of the error values described in the “CSTAUniversalFailureConfEvent” section in Chapter 3:

- GENERIC_OPERATION_REJECTION (71) If the application has not registered to receive System Status Event reports, the cstaChangeSysStatFilter() request will be rejected.

Detailed Information:

- The linkStatReq private parameter is only useful in multilink configurations.
- If the application requests System Status Event Reports for changes in specific CTI link states (up/down/unavailable), they must examine the private data included in the CSTASysStatEvent event report to determine the changes in the individual CTI links.
- The count and plinkStatus private ack parameters will only be provided when the linkStatReq parameter was set to TRUE in the Change System Status Start service request.
- If the linkStatReq private parameter was not specified, there will be no changes in the reporting of System Status events for changes in the state of individual G3 CTI links.

For more information, refer to “System Status Event” in this chapter.

Syntax

```

#include <acs.h>
#include <csta.h>

// cstaChangeSysStatFilter() - Service Request

RetCode_t  cstaChangeSysStatFilter (
            ACSHandle_t          acsHandle,
            InvokeID_t           invokeID,
            SystemStatusFilter_t statusFilter,
            PrivateData_t        *privateData);

typedef unsigned char          SystemStatusFilter_t;

#define          SF_INITIALIZING          0x80
#define          SF_ENABLED              0x40
#define          SF_NORMAL               0x20
#define          SF_MESSAGES_LOST       0x10
#define          SF_DISABLED             0x08
#define          SF_OVERLOAD_IMMINENT   0x04
#define          SF_OVERLOAD_REACHED    0x02
#define          SF_OVERLOAD_RELIEVED   0x01

// CSTAChangeSysStatFilterConfEvent - Service Response

typedef struct
{
    ACSHandle_t  acsHandle;
    EventClass_t eventClass; // CSTACONFIRMATION
    EventType_t  eventType;  // CSTA_CHANGE_SYS_STAT_FILTER_CONF
} ACSEventHeader_t;

typedef struct CSTAChangeSysStatFilterConfEvent_t {
    SystemStatusFilter_t  statusFilterSelected;
    SystemStatusFilter_t  statusFilterActive;
} CSTAChangeSysStatFilterConfEvent_t;

typedef struct
{
    ACSEventHeader_t eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
            union
            {
                CSTAChangeSysStatFilterConfEvent_t changeSysStatFilter;
            } u;
        } cstaConfirmation;
    } event;
    char  heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

```

Private Data Version 5 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attSysStat() - Service Request Private Data Setup Function

RetCode_t    attSysStat(
    ATTPrivateData_t*attPrivateData, // length must be set
    Boolean    linkStatusReq); // send event reports for
                                // CTI link state changes

typedef struct ATTPrivateData_t
{
    char                vendor[32];
    unsigned short     length;
    char                data[ATT_MAX_PRIVATE_DATA];
}

// ATTLinkStatusEvent - Service Response Private Data

typedef struct
{
    ATTEventType_teventType;// ATT_LINK_STATUS
    union
    {
        ATTLinkStatusEvent_tlinkStatus;
    } u;
} ATTEvent_t;

typedef struct ATTLinkStatusEvent_t
{
    short                count;
    ATTLinkStatus_t     *pLinkStatus;
} ATTLinkStatusEvent_t;

typedef struct ATTLinkStatus_t
{
    short                linkID;
    ATTLinkState_t      linkState;
} ATTLinkStatus_t;

typedef enum ATTLinkState_t {
    LS_LINK_UNAVAIL= 0,    // the link is disabled
    LS_LINK_UP    = 1,    // the link is up
    LS_LINK_DOWN= 2      // the link is down
} ATTLinkState_t;
```


Private Data Version 4 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attSysStat() - Service Request Private Data Setup Function

RetCode_t    attSysStat(
    ATTPrivateData_t*attPrivateData, // length must be set
    Boolean    linkStatusReq); // send event reports for
                                // CTI link state changes

typedef struct ATTPrivateData_t
{
    char                vendor[32];
    unsigned short     length;
    char                data[ATT_MAX_PRIVATE_DATA];
}

// ATTV4LinkStatusEvent - Service Response Private Data

typedef struct
{
    ATTEventType_teventType;// ATTV4_LINK_STATUS
    union
    {
        ATTV4LinkStatusEvent_tv4linkStatus;
    } u;
} ATTEvent_t;

typedef struct ATTV4LinkStatusEvent_t
{
    short                count;
    ATTLinkStatus_t     linkStatus[8];
} ATTV4LinkStatusEvent_t;

typedef struct ATTLinkStatus_t
{
    short                linkID;
    ATTLinkState_t      linkState;
} ATTLinkStatus_t;

typedef enum ATTLinkState_t {
    LS_LINK_UNAVAIL= 0,    // the link is disabled
    LS_LINK_UP    = 1,    // the link is up
    LS_LINK_DOWN= 2      // the link is down
} ATTLinkState_t;
```

Private Data Versions 2 and 3 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// attSysStat() - Service Request Private Data Setup Function

RetCode_t attSysStat(
    ATTPrivateData_t*attPrivateData, // length must be set
    Boolean linkStatusReq); // send event reports for
    // CTI link state changes

typedef struct ATTPrivateData_t
{
    char vendor[32];
    unsigned short length;
    char data[ATT_MAX_PRIVATE_DATA];
}

// ATTV3LinkStatusEvent - Service Response Private Data

typedef struct
{
    ATTEventType_teventType;// ATTV3_LINK_STATUS
    union
    {
        ATTV3LinkStatusEvent_tv3linkStatus;
    } u;
} ATTEvent_t;

typedef struct ATTV3LinkStatusEvent_t
{
    short count;
    ATTLinkStatus_t linkStatus[4];
} ATTV3LinkStatusEvent_t;

typedef struct ATTLinkStatus_t
{
    short linkID;
    ATTLinkState_t linkState;
} ATTLinkStatus_t;

typedef enum ATTLinkState_t {
    LS_LINK_UNAVAIL= 0, // the link is disabled
    LS_LINK_UP = 1, // the link is up
    LS_LINK_DOWN= 2 // the link is down
} ATTLinkState_t;
```

System Status Event

Direction: Switch to Client

Event: CSTASysStatEvent

Service Parameters: systemStatus

Private Parameters: count, plinkStatus (private data version 5), linkStatus (private data versions 2, 3, and 4)

Functional Description:

This unsolicited event is sent by the G3 driver to inform the application of changes in system status. The application must have previously registered to receive System Status Events via the cstaSysStatStart() service request. The System Status Event Reports will be sent for those events that have not been filtered by the application via the cstaSysStatStart() and cstaChangeSysStatFilter() service requests.

Service Parameters:

- systemStatus** [mandatory — partially supported] This parameter contains a value that identifies the change in overall system status detected by the G3 PBX driver. The following System Status events will be sent to the application by the G3 driver/switch if the application has not filtered the event:
- **SS_ENABLED** — A CSTASysStatEvent event report will be sent with the systemStatus set to SS_ENABLED when the first CTI link to the G3 switch has been established from an “all CTI links down” state. The application can examine the private data portion of the event report to determine which CTI links are up (LS_LINK_UP), which CTI links are down (LS_LINK_DOWN), and which CTI links are disabled via the OA&M interface (LS_LINK_UNAVAIL). No Call or Device Monitors, or Routing Sessions should exist at this point.
 - **SS_DISABLED** — A CSTASysStatEvent event report will be sent with the systemStatus set to SS_DISABLED when the last CTI link to the G3 switch has failed. The application can examine the private data portion of the event report, but it will always indicate that all CTI links are down (LS_LINK_DOWN) or unavailable (LS_LINK_UNAVAILABLE). All Call and Device Monitors will be terminated, all Routing Sessions will be aborted, and all outstanding CSTA requests should be negatively acknowledged.
 - **SS_NORMAL**— A CSTASysStatEvent event report will be sent with the systemStatus set to SS_NORMAL when the application has requested event reports for changes in specific CTI link states (via the linkStatusReq private parameter in the cstaSysStatStart() or cstaChangeSysStatFilter()) and a CTI link changes state to up, (LS_LINK_UP) down (LS_LINK_DOWN), or unavailable/busied-out via OA&M (LS_LINK_UNAVAIL). The systemStatus normal (SS_NORMAL) indicates that at least one CTI link to the switch is available. The application can examine the private data portion of the event report to determine which CTI links are up, down, or unavailable/busied-out. Call or Device Monitors, and Routing Sessions may have been terminated when the CTI link state changed to down (LS_LINK_DOWN).

Private Parameters:

count Identifies the number of CTI links described in the plinkStatus private ack parameter. This parameter is only provided when the linkStatusReq private parameter was set to TRUE.

plinkStatus Specifies the status of each CTI link to the switch. This parameter is only provided when the linkStatusReq private parameter was set to TRUE. The plinkStatus private data parameter will indicate the availability of each administered CTI link to the G3 switch to which the application is connected.

The status of each link identified by linkID will be set to one of the following values in the linkState field:

- LS_LINK_UP — The link is able to support telephony services to the switch.
- LS_LINK_DOWN — The link is unable to support telephony services to the switch.
- LS_LINK_UNAVAIL —The link has been disabled (busied-out) via the OA&M interface and will not support new CSTA requests. Existing telephony service requests maintained by this link will continue.

This parameter is supported by private data version 5 and later only.

linkStatus Specifies the status of each CTI link to the switch. For details, see the description for the plinkStatus private ack parameter. This parameter is supported by private data versions 2, 3, and 4.

Detailed Information:

- If multiple CTI links are connected and administered to a specific switch, the systemStatus parameter will indicate the aggregate link status. When the first CTI link is established from an “all CTI links down” state, a System Status Event Report will be sent to the application with the systemStatus set to SS_ENABLED. When the last CTI fails (a transition to the “all CTI links down” state), a System Status Event Report will be sent to the application with the systemStatus set to SS_DISABLED.
- If multiple CTI links are connected and administered to a specific switch, Private Data must be used to determine if the switching system is performing as administered. The plinkStatus private parameter can be used to check the status of each individual CTI link.

Syntax

```
#include <acs.h>
#include <csta.h>

// CSTASysStatEvent - System Status Event

typedef struct
{
    ACSHandle_t acsHandle;
    EventClass_t eventClass; // CSTAEVENTREPORT
    EventType_t eventType; // CSTA_SYS_STAT
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t eventHeader;

    union
    {
        struct
        {
            union
            {
                CSTASysStatEvent_t sysStat;
            } u;
        } cstaEventReport;
    } event;
    char heap[CSTA_MAX_HEAP];
} CSTAEvent_t;

typedef struct CSTASysStatEvent_t {
    SystemStatus_t systemStatus;
} CSTASysStatEvent_t;

typedef enum SystemStatus_t {
    SS_INITIALIZING = 0, // Not supported
    SS_ENABLED = 1, // Supported
    SS_NORMAL = 2, // Supported
    SS_MESSAGES_LOST = 3, // Not supported
    SS_DISABLED = 4, // Supported
    SS_OVERLOAD_IMMINENT = 5, // Not supported
    SS_OVERLOAD_REACHED = 6, // Not supported
    SS_OVERLOAD_RELIEVED = 7 // Not supported
} SystemStatus_t;
```

Private Data Version 5 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTLinkStatusEvent - System Status Event Private Data

typedef struct
{
    ATTEventType_t eventType; // ATT_LINK_STATUS
    union
    {
        ATTLinkStatusEvent_t linkStatus;
    } u;
} ATTEvent_t;

typedef struct ATTLinkStatusEvent_t
{
    short count;
    ATTLinkStatus_t *pLinkStatus;
} ATTLinkStatusEvent_t;

typedef struct ATTLinkStatus_t
{
    short linkID;
    ATTLinkState_t linkState;
} ATTLinkStatus_t;

typedef enum ATTLinkState_t {
    LS_LINK_UNAVAIL= 0, // the link is disabled
    LS_LINK_UP = 1, // the link is up
    LS_LINK_DOWN= 2 // the link is down
} ATTLinkState_t;
```

Private Data Version 4 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV4LinkStatusEvent - System Status Event Private Data

typedef struct
{
    ATTEventType_t eventType; // ATTV4_LINK_STATUS
    union
    {
        ATTV4LinkStatusEvent_tv4linkStatus;
    } u;
} ATTEvent_t;

typedef struct ATTV4LinkStatusEvent_t
{
    short count;
    ATTLINKSTATUS_t linkStatus[8];
} ATTV4LinkStatusEvent_t;

typedef struct ATTLINKSTATUS_t
{
    short linkID;
    ATTLINKSTATE_t linkState;
} ATTLINKSTATUS_t;

typedef enum ATTLINKSTATE_t {
    LS_LINK_UNAVAIL= 0, // the link is disabled
    LS_LINK_UP = 1, // the link is up
    LS_LINK_DOWN= 2 // the link is down
} ATTLINKSTATE_t;
```


Private Data Versions 2 and 3 Syntax

```
#include <acs.h>
#include <csta.h>
#include <attpriv.h>

// ATTV3LinkStatusEvent - System Status Event Private Data

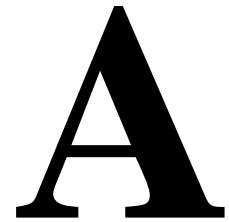
typedef struct
{
    ATTEventType_t eventType; // ATTV3_LINK_STATUS
    union
    {
        ATTV3LinkStatusEvent_t tv3linkStatus;
    } u;
} ATTEvent_t;

typedef struct ATTV3LinkStatusEvent_t
{
    short count;
    ATTLinkStatus_t linkStatus[4];
} ATTV3LinkStatusEvent_t;

typedef struct ATTLinkStatus_t
{
    short linkID;
    ATTLinkState_t linkState;
} ATTLinkStatus_t;

typedef enum ATTLinkState_t {
    LS_LINK_UNAVAIL= 0, // the link is disabled
    LS_LINK_UP = 1, // the link is up
    LS_LINK_DOWN= 2 // the link is down
} ATTLinkState_t;
```

Enhanced Voice Terminal Display



Feature Description

The following details on the Enhanced Voice Terminal Display feature are taken from the *DEFINITY Enterprise Communication Server Feature Description*, Release 5.

Considerations

After the feature is turned on at the system parameter level, you must still enter the display field values on the appropriate forms and submit those forms before the enhanced character set will display. The reverse is also true, in that if you turn the feature off at the system level after Group 2 display has been administered, you must change the display field values back to the appropriate Group 1 characters.

Interactions

Internal Feature interactions

- Directory

The Enhanced Voice Terminal Display feature will interact with the Directory feature to include names administered with enhanced display characters in the Directory. Terminal users can select directory entries displayed in the Group 2 character set by using the “Directory” and “Next” buttons on their display terminals, and can use the “*” to indicate special characters in a directory search. However, they cannot input directory entries using the voice terminal because of the special characters used.

- ISDN
Enhanced display characters are supported for ISDN calls, and will display correctly for calls between properly administered DEFINITY switches with the same terminal types at both ends.
- Data Call Setup Dialing
Not supported.
- Message Retrieval - Print Messages (Demand Print)
Not supported.
- Leave Word Calling - Adjunct
Not supported.
- OSSI
Displays the literal value of the display field, not the Group 2 characters.
- Features with Display Interactions
All features that display administered data are supported for enhanced display, provided the terminal hardware is compatible.

External Feature Interfaces

- Adjunct Switch Applications Interface (ASAI)
The information sent from the DEFINITY ECS to any adjunct is the literal value of the field as administered, not the enhanced display format. Since the adjunct has no way to interpret this into special fonts, the resulting display appears as a string of random characters, for example, as "2<@^."
The ASAI-Query Names Database feature also will also receive names as mentioned above. The same is true of the ASAI-Accessed BCMS Data feature.
- AUDIX
The following AUDIX products operate normally with Enhanced Voice Terminal Display enabled:
 - Basic R1 AUDIX
 - Embedded DCIU Audix
 - Embedded DCP Audix (requires R3.2 or greater)
- AUDIX Voice Power/Audix Voice Power Lodging
Not supported.
- CentreVu Direct Connect
Not supported.

- Novell Telephony Services
Not supported.
- Look Ahead Interflow
This feature can use the enhanced character set, but terminal types must match the administered display group.
- VUStats
If this feature is used with non-supported terminals, the enhanced character set may cause screens to be cleared or display false information.
- DCS Networking
This feature is supported in a DCS environment. All switches in the network must have the enhanced character set enabled *and* the same software load must be installed on each server.
- Monitor 1 and One Vision
Monitor 1 and One Vision will receive the ASCII characters as administered.
- ECMA QSIG Networking
If two DEFINITY servers are administered for enhanced display for terminals using the same character group, this feature is supported. It is not supported between a DEFINITY ECS and a non-DEFINITY switch.

Administration

System administrators can activate this feature by setting the value of the Enhanced 84XX Display Character Set? field on the System-Parameters Country-Options form to **y**. Once this value is set, the administrator can enter the Roman characters that produce the corresponding Group 2 characters on the terminal display.

This feature is activated on a system-wide basis using the System-Parameters County-Options form. Once activated, the administrator must update those forms that determine the contents of display fields. Users can administer name fields and soft key labels to display in the character set of choice.

The following forms contain fields that will accept the enhanced display characters:

- Access-endpoint
- Agent-loginID
- Announcements
- Attendant
- Bcms-vustats loginIDs

- Console-parameters
- Data-module
- Display-messages
- Display-messages auto-wakeup-dn-dst
- Display-messages call-identifiers
- Display-messages date-time
- Display-messages leave-word-calling
- Display-messages malicious-call-trace
- Display-messages miscellaneous-features
- Display-messages property-management
- Display-messages softkey-labels
- Display-messages time-of-day-routing
- Display-messages vustats
- Hunt-group
- Listed-directory-numbers
- Paging loudspeaker
- Personal-CO-line
- Pri-endpoint
- Station
- System-parameters-hospitality
- Term-ext-group
- Trunk-group
- Vdn
- Vustats-display-format

Implementation of Enhanced Voice Terminal Display

The sections below describe how to implement the Enhanced Voice Terminal Display feature. For additional information, refer to *DEFINITY Enterprise Communications Server Implementation*, Volume 1 or 2, Release 5.

Description

The Enhanced Voice Terminal Display feature allows users to administer the switch to display information and soft key labels on display terminals, using either the Katakana alphabet or various European characters, In addition to the Roman alphabet. The system accepts entries as Roman characters preceded by a tilde (~), and maps them to corresponding Katakana or European characters.

The character set that displays on supported voice terminals can be divided into two different groups.

- Group 1 contains the Roman alphabet, numerals and special characters found on the standard US English keyboard.
- Group 2 contains one of two alternate character sets:
 - Group 2a contains Katakana characters, as well as some European characters and other symbols.
 - Group 2b contains characters required to display most European languages.

The enhanced display feature allows users to administer the system to display either Group 1 or one of the Group 2 character sets. Whether your system displays group 2a or 2b depends on the terminal types attached to the ECS (see the table below).

Administration

To administer voice terminal displays in Group 2 characters, first update the following form.

Form	Field	Form Instructions (Page)
System-Parameters Country-Options	Enhanced 84xx Terminal Display?	5-300

Hardware Requirements

Each character set for Enhanced Voice Terminal Display requires a specific type of terminal, based on which Group 2 character set you want to display.

⇒ NOTE:

It is very important to make sure that you use the same terminal type across your entire system, that is, use only Group 2a or Group 2b terminals. Otherwise, it is impossible to guarantee that the displays will appear as indicated. Your Lucent Technologies representative can make sure you have the appropriate terminal types.

Feature Implementation

Execute the following steps to implement this feature:

1. Activate the feature by changing the value of the Enhanced 84xx Display Character Set? field to "y" on the System-Parameters Country-Options form.
2. Enter the appropriate values in the display fields of the forms that you want to use. See "Update display fields" below for detailed procedures.
3. Submit all forms after you modify them.

⇒ NOTE:

To display Roman characters once you have activated this feature, simply enter the values of the display fields as you normally would.

```

                                SYSTEM PARAMETERS COUNTRY-OPTIONS

    Companding Mode:  Mu Law           Base Tone Generation Set:  1
    440Hz PBX-dial Tone?  n           440Hz Secondary-dial Tone?  n
    Digital Loss Plan:   1             Version of Digital Loss Plan:  1
    Analog Ringing Cadence: 1         Set Layer 1 timer T1 to 30 seconds?  n
    Analog Line Transmission: 1       Enhanced 84xx Display Character Set?  n

TONE DETECTION PARAMETERS
    Tone Detection Mode:  6             Dial Tone Validation Timer:
    Interdigit Pause:   short

```

Screen A-1. The System Parameters Country Options form

Update display fields

Once you have activated the feature, you can add or update the fields that you want to display using the Group 2 character set. To do this:

1. Retrieve the appropriate form.
2. Move to the display field you want to add or change.
3. At the point where you want to start the display in Group 2 characters, enter a tilde (~).
4. Use the character map on the following pages to determine the Roman characters that correspond to the Group 2 characters you want.
5. Type the Roman characters in the display field.
6. If you want to end the display of Group 2 characters and continue with Roman characters, type another tilde, then continue using the Roman set. If you want the entire field to display in Group 2 characters, you can omit the tilde from the end of the field.
7. Submit the form.

NOTE:

The Katakana characters that combine symbols do not appear as single characters in the terminal display set. To display these characters, you must use a combination. See the example below.

Also, the characters will appear on the display terminal in the order that you enter them. If you want the display to read right to left, you must enter the characters in reverse order on the form.

Examples

Table A-1 shows a few examples of what you would need to enter in a display value field in order for the Group 2 characters to appear.

Table A-1. Characters to Enter for Group 2 Character Display

Enter this:	To display this:
~2<@^	インダ
~2<@^~, Max	インダ, Max
~N_	ホ
Pe~n~a	Peña
Fr~a~ulein	Fräulein

Table A-2. Character Map, Group 1 to Group 2a

Group 2a	Group 1	Group 2a	Group 1
space	space	ク	8
。	!	ケ	9
「	“	コ	:
」	#	サ	;
、	\$	シ	<
•	%	ス	=
ヲ	&	セ	>
ア	'	ソ	?
イ	(タ	@
ウ)	チ	A
エ	*	ツ	B
オ	+	テ	C
ヤ	,	ト	D
ユ	-	ナ	E
ヨ	.	ニ	F
ッ	/	ヌ	G
ー	0	ネ	H
ア	1	ノ	I
イ	2	ハ	J
ウ	3	ヒ	K
エ	4	フ	L
オ	5	ヘ	M
カ	6	ホ	N
キ	7	マ	O

Table A-3. Character Map, Group 1 to Group 2a, continued

Group 2a	Group 1	Group 2a	Group 1
ミ	P	μ	d
ム	Q	σ	e
メ	R	ρ	f
モ	S	√	h
ヤ	T	-1	i
ユ	U	*	k
ヨ	V	∅	l
ラ	W	£	m
リ	X	ñ	n
ル	Y	ö	o
レ	Z	θ	r
ロ	[∞	s
ワ	\	Ω	t
ン]	ü	u
ゝ	^	Σ	v
◦	—	π	w
α	‘	\bar{x}	x
ä	a	÷	}
β	b	¥	

**NOTE:**

For Group 2a, the z and { characters map to Kanji characters as follows:

z - symbol for 1,000

{ - symbol for 10,000

Group 2b character map

The following pages contain the character map for Group 2b.

⇒ NOTE:

Some of the characters in the following map appear in only upper or lower case, for example, *ó*, *ć*, *ž*, and others. These display the same for both upper and lower case.

Table A-4. Character Map, Group 1 to Group 2b

Group 2b	Group 1	Group 2b	Group 1
í	space	È	8
ï	!	č	9
ã	“	Ł	:
á	#	đ	;
à	\$	Ý	<
ú	%	ø	=
ù	&	æ	>
é	'	Ó	?
è	(î	@
é)	å	A
ł	*	ą	B
Ď	+	â	C
ý	,	ů	D
ž	-	û	E
ő	.	ê	F
ò	/	é	G
Í	0	ę	H
Ī	1	Ç	I
Ã	2	ř	J
Á	3	Đ	K
À	4	ÿ	L
Ú	5	ń	M
Ù	6	Æ	N
É	7	õ	O

Table A-5. Character Map, Group 1 to Group 2b, continued

Group 2b	Group 1	Group 2b	Group 1
Î	P	ğ	h
Å	Q	Š	i
Ȧ	R	Ŧ	j
Â	S	ı	k
Ů	T	ž	l
Û	U	ň	m
Ê	V	ñ	n
È	W	ö	o
Ě	X	Ì	p
Š	Y	Ä	q
Ř	Z	ă	r
´	[Ă	s
ÿ	\	Û	t
Ň]	ü	u
Ñ	^	Ě	v
Ô	_	Ě	w
ì	`	Ğ	x
ä	a	ş	y
ß	b	Ť	z
û	d	þ	{
Ü	e	ž	
ě	f	Ň	}
ë	g		

Table A-6. Troubleshooting tips

Problem	Cause/Solution
The characters that display are not what you thought you entered.	This feature is case sensitive. Check the table to make sure you entered the right case.
You entered a lower case “c”, and “**” appears on the display instead.	The lower case “c” has a specific meaning in the DEFINITY system, and therefore cannot be mapped to any other character. An asterisk “**” appears in its place.
You entered “->” or “<-” and nothing appears on the display.	These characters do not exist as single keys on the standard US-English keyboard. Therefore the system is not programmed to handle them.
Enhanced display characters appear in fields that you did not update.	If an existing display field contains a tilde (~) followed by Roman characters, and you update and submit that form after this feature is activated, that field will display the enhanced character set.
Nothing displays on the terminal at all.	Some non-supported terminals do not display anything if a special character is presented. Check the model of display terminal you are using.
You entered a character with a descender and part of it appears cut off in the display.	Some of the characters in Group 2a have descenders that do not appear entirely within the display area. These characters are not included in the character map. For these characters (g,j,p,q,y), use Group 1 equivalents.

Index

Symbols

- * and # characters
 - send DTMF tone, 4-132

A

- AAR/ARS
 - make call, 4-74
- Abbreviated dialing
 - originated event, 9-131
- Account codes
 - originated event, 9-131
- ACD destination
 - make call, 4-74
- ACD group
 - device type, 3-28
- ACD originator
 - make call, 4-74
- ACD split
 - monitor calls via device, 8-25
 - monitor device, 8-34
- Ack parameters
 - alternate call, 4-10
 - answer call, 4-14
 - change monitor filter, 8-7
 - change system status filter, 11-23
 - clear call, 4-18
 - clear connection, 4-21
 - conference call, 4-28
 - consultation call, 4-36
 - consultation direct-agent call, 4-45
 - consultation supervisor-assist call, 4-53
 - conventions, 3-45
 - deflect call, 4-61
 - hold call, 4-66
 - make call, 4-73
 - make direct-agent call, 4-84
 - make predictive call, 4-95
 - make supervisor-assist call, 4-105
 - monitor call, 8-15
 - monitor calls via device, 8-24
 - monitor device, 8-33
 - monitor stop, 8-46
 - monitor stop on call, 8-42
 - pickup call, 4-114
 - query agent login, 6-7
 - query agent state, 6-14
 - reconnect call, 4-120
 - retrieve call, 4-127
 - route end service (TSAPI v2), 10-7
 - route register, 10-18
 - route register cancel, 10-14
 - route request (TSAPI v2), 10-24
 - route select (TSAPI v2), 10-43
 - selective listening hold, 4-138
 - selective listening retrieve, 4-144
 - send DTMF tone, 4-131, 4-132
 - set advice of charge, 5-3
 - set agent state, 5-11
 - set billing rate, 5-19
 - set do not disturb feature, 5-23
 - set forwarding feature, 5-27
 - set MWI feature, 5-31
 - single step conference call, 4-151
 - system status request, 11-4
 - system status start, 11-11
 - system status stop, 11-19
 - transfer call, 4-158
- Ack private parameters
 - change monitor filter, 8-8
 - change system status filter, 11-23
 - conference call, 4-28
 - consultation call, 4-36
 - consultation direct-agent call, 4-45
 - consultation supervisor-assist call, 4-54
 - conventions, 3-45
 - make call, 4-73
 - make direct-agent call, 4-85
 - make predictive call, 4-95
 - make supervisor-assist call, 4-106
 - monitor call, 8-15
 - monitor calls via device, 8-24
 - monitor device, 8-33
 - monitor stop on call, 8-42
 - query ACD split, 6-3
 - query agent login, 6-7
 - query agent state, 6-14
 - set advice of charge, 5-3
 - set agent state, 5-11
 - single step conference call, 4-151
 - system status request, 11-4
 - system status start, 11-12
 - transfer call, 4-158
- ACS parameter syntax, 3-48
- ACS stream
 - set advice of charge, 5-3
- Activation
 - set forwarding feature, 5-28
- Active call
 - reconnect call, 4-127
- Active state
 - retrieve call, 4-127
- Adjunct messages
 - set MWI feature, 5-31
- Adjunct-controlled splits
 - monitor calls via device, 8-25
- Administration without hardware
 - deflect call, 4-62
 - monitor device, 8-34
 - pickup call, 4-115

- Advice of charge event report
 - monitor call, 8-17
- Agent activity
 - mapped to agent state, 6-16
 - mapped to talk state, 6-16
- Agent event filters, 8-4
- Agent state
 - mapped to agent activity, 6-16
- Agent state, mapped to agent work mode, 6-16
- Agent work mode, mapped to agent state, 6-16
- AgentMode service parameter, 5-12
- Alternate call
 - ack parameters, 4-10
 - description, 4-9
 - detailed information, 4-11
 - overview, 4-2
 - service parameters, 4-10
 - syntax, 4-12
- Analog ports
 - monitor device, 8-34
- Analog sets, 9-165
- Analog station operation
 - alternate call, 4-15
 - answer call, 4-15
 - reconnect call, 4-15
- Analog stations
 - alternate call, 4-67
 - clear connection, 4-22
 - conference call, 4-30
 - consultation call, 4-67
 - hold call, 4-67
 - make call, 4-74
 - reconnect call, 4-22
 - transfer call, 4-159
- ANI screen pop application requirements, 9-166
- Announcement destination
 - make call, 4-75
- Announcements, 9-167, 9-169
 - selective listening hold, 4-139
 - selective listening retrieve, 4-139
- Answer call, 4-13
 - ack parameters, 4-14
 - analog station operation, 4-15
 - detailed information, 4-14
 - nak parameters, 4-14
 - overview, 4-2
 - service parameters, 4-14
 - syntax, 4-17
- Answer supervision timeout, 9-167
- Applications
 - designing using original call info, 3-9
 - designing with private data, 3-13
 - designing, with screen pop information, 3-7
 - migration from private data v5 to v6, 3-19
 - remote, passing UUI, 3-10
- AT&T MultiQuest 900 Vari-A-Bill, 5-18
- Attendant auto-manual splitting, 9-168
- Attendant call waiting, 9-168
- Attendant control of trunk group access, 9-169

- Attendant groups, 9-167
 - monitor device, 8-34
- Attendant specific button operation, 9-168
- Attendants, 9-167
 - deflect call, 4-62
 - make call, 4-75
 - monitor device, 8-34
 - pickup call, 4-115
 - selective listening hold, 4-139
 - selective listening retrieve, 4-139
- Attributes, of parameters, 3-45
- AUDIX, 9-169
 - send DTMF tone, 4-132
- Authorization codes
 - make call, 4-75
 - originated event, 9-131
- Auto call back
 - deflect call, 4-62
 - pickup call, 4-115
- Auto-available split, 9-170
- Automatic Call Distribution (ACD), 9-167, 9-169
- Automatic callback
 - originated event, 9-131

B

- Blind transfer
 - established event, 9-90
- Bridged call appearance, 9-170
 - alternate call, 4-67
 - clear connection, 4-22
 - conference call, 4-30
 - consultation call, 4-67
 - deflect call, 4-62
 - hold call, 4-67
 - make call, 4-75
 - originated event, 9-131
 - pickup call, 4-115
 - reconnect call, 4-22, 4-128
 - retrieve call, 4-128
 - single step conference call, 4-153
 - transfer call, 4-160
- Bridged state, 7-8
- Busy Hour Call Completions (BHCC), 5-3
- Busy verification of terminals, 9-171
 - alternate call, 4-67
 - consultation call, 4-67
 - hold call, 4-67

C

- Call appearance button, 9-168
- Call classification
 - established event, 9-89

- make call, 4-75
- Call cleared event
 - description, 9-3
 - detailed information, 9-5
 - monitor device, 8-31
 - private parameter syntax, 9-7
 - private parameters, 9-4
 - redirection on no answer, 9-165
 - report, 9-3, 9-32
 - service parameters, 9-4
 - syntax, 9-6
- Call clearing state
 - charge advice event, 9-10
- Call control service group
 - supported services, 3-2
 - unsupported services, 3-5
- Call coverage, 9-171
- Call coverage path containing VDNs, 9-172
 - make call, 4-75
- Call delivered
 - to ACD device, 9-40
 - to ACD split, 9-41
 - to station device, 9-39
 - to VDN, 9-40
- Call destination
 - make call, 4-75
- Call event filters, 8-3
- Call event reports
 - Monitor stop on call, 8-43
- Call forwarding
 - pickup call, 4-115
- Call forwarding all calls, 9-172
 - make call, 4-75
 - set forwarding feature, 5-27
- Call identifier, 3-35
 - syntax, 3-35
- Call monitoring event sequences
 - single step conference call, 4-153
- Call objects, 3-35
- Call originator type, 9-56
- Call park, 9-172
 - originated event, 9-131
- Call pickup, 9-173
- Call prompting, 9-175
 - for screen pop, 3-7
- Call state, 3-35
 - send DTMF tone, 4-132
 - single step conference call, 4-153
- Call states, 7-8
- Call vectoring, 9-173
 - selective listening hold, 4-139
 - selective listening retrieve, 4-139
- Call vectoring, interactions with feedback, 9-173
- Call waiting, 9-175
 - deflect call, 4-62
 - pickup call, 4-115
- Called number
 - for screen pop, 3-7
- Calling number
 - for screen pop, 3-7
- Calls
 - phantom, 3-28
- Calls In queue, number, 9-177
- Cancel button, 9-168
- Cancel requested service, 8-3
- Capacity, system, 3-40
- Change monitor filter, 8-1
 - ack parameters, 8-7
 - ack private parameters, 8-8
 - description, 8-6
 - detailed information, 8-8
 - nak parameters, 8-8
 - private data v2-4 syntax, 8-12
 - private data v5 syntax, 8-11
 - private parameters, 8-7
 - service parameters, 8-7
 - syntax, 8-9
- Change system status filter
 - ack parameters, 11-23
 - ack private parameters, 11-23
 - description, 11-21
 - detailed information, 11-24
 - nak parameters, 11-24
 - overview, 11-2
 - private data v2-3 syntax, 11-28
 - private data v4 syntax, 11-27
 - private data v5 syntax, 11-26
 - private parameters, 11-22
 - service parameters, 11-22
 - syntax, 11-25
- Charge advice event
 - description, 9-8
 - detailed information, 9-10
 - private parameter syntax, 9-12
 - private parameters, 9-9
 - report, 9-8
 - service parameters, 9-9
 - syntax, 9-11
- Charge advice events, 8-31
- Class of Restrictions (COR)
 - make call, 4-75
- Class of Service (COS)
 - make call, 4-76
- Clear call
 - ack parameters, 4-18
 - description, 4-18
 - detailed information, 4-18
 - nak parameters, 4-18
 - overview, 4-3
 - service parameters, 4-18
 - syntax, 4-19
- Clear connection
 - ack parameters, 4-21
 - description, 4-20
 - detailed information, 4-22
 - nak parameters, 4-22

- overview, 4-3
 - private data v2-5 syntax, 4-26
 - private data v6 syntax, 4-25
 - private parameters, 4-21
 - service parameters, 4-21
 - syntax, 4-24
 - userInfo parameter, 4-21
- Conference, 9-176
- Conference call
- ack parameters, 4-28
 - ack private parameters, 4-28
 - detailed information, 4-30
 - nak parameters, 4-29
 - overview, 4-3
 - private data v5 syntax, 4-32
 - selective listening hold, 4-139
 - selective listening retrieve, 4-139
 - service parameters, 4-28
 - syntax, 4-31
- Conference event
- report, 9-13
- Conferenced event
- description, 9-13
 - detailed information, 9-18
 - private data v2-3 syntax, 9-29
 - private data v4 syntax, 9-26
 - private data v5 syntax, 9-23
 - private data v6 syntax, 9-20
 - private parameters, 9-16
 - service parameters, 9-14
 - syntax, 9-19
 - trunkList parameter, 9-17
 - userInfo parameter, 9-17
- Conferencing call, with screen pop information, 3-7
- Conferencing calls
- CSTA services used, 3-8
- Confirmation event
- format, 3-45
- Confirmation interface structures
- private data v4 syntax, 3-15
 - private data v5-6 syntax, 3-14
- Connection cleared event
- description, 9-32
 - detailed information, 9-35
 - private data v2-5 syntax, 9-38
 - private parameter syntax, 9-37
 - private parameters, 9-35
 - report, 9-32
 - service parameters, 9-34
 - syntax, 9-36
 - userInfo parameter, 9-35
- Connection identifier, 3-36
- syntax, 3-36
- Connection identifier conflict, 3-36
- Connection object, 3-35
- Connection state, 3-37
- send DTMF tone, 4-132
 - syntax, 3-39
- Connection state definitions, 3-38
- Consult, 9-176
- Consultation call
- ack parameters, 4-36
 - ack private parameters, 4-36
 - description, 4-33
 - detailed information, 4-37, 4-96
 - nak parameters, 4-36
 - overview, 4-4
 - private data v2-5 syntax, 4-41
 - private data v6 syntax, 4-39
 - private parameters, 4-35
 - service parameters, 4-34
 - syntax, 4-38
 - userInfo parameter, 4-35
- Consultation direct-agent call
- ack parameters, 4-45
 - ack private parameters, 4-45
 - description, 4-42
 - detailed information, 4-46
 - nak parameters, 4-45
 - overview, 4-4
 - private data v2-5 syntax, 4-50
 - private data v6 syntax, 4-48
 - private parameters, 4-44
 - service parameters, 4-43
 - syntax, 4-47
 - userInfo parameter, 4-44
- Consultation supervisor-assist call
- ack parameters, 4-53
 - ack private parameters, 4-54
 - description, 4-51
 - detailed information, 4-55
 - nak parameters, 4-54
 - overview, 4-5
 - private data v2-5 syntax, 4-59
 - private data v6 syntax, 4-57
 - private parameters, 4-53
 - service parameters, 4-52
 - syntax, 4-56
 - userInfo parameter, 4-53
- Consultation transfer
- established event, 9-90
- Conventions
- ack parameters, 3-45
 - ack private parameters, 3-45
 - confirmation event, 3-45
 - for G3 CSTA services, 3-45
 - for private data, 3-19
 - format, 3-45
 - function, 3-45
 - functional description, 3-45
 - nak parameters, 3-45
 - private data, 3-45
 - private parameters, 3-45
 - service parameters, 3-45
- Converse agent
- selective listening hold, 4-140
 - selective listening retrieve, 4-140
- Cover all

- pickup call, 4-115
- CSTA local call state, mapped to G3 local call state, 7-8
- CSTA objects
 - call, 3-35
 - device, 3-28
 - device type, 3-28
- CSTA services
 - snapshot call, 7-2
 - snapshot device, 7-6
 - supported, 3-2
 - unsupported, 3-5
- cstaDeflectCall
 - pickup call, 4-115
- CSTAEventCause, definitions, 9-1
- cstaMakePredictiveCall
 - originated event, 9-131
- CSTAUniversalFailureConfEvent, 3-49
- CTI link failure, 9-176
- CTI links
 - multiple, considerations for, 3-43
- Customer support
 - for DEFINITY G3 PBX Driver, 1-6
 - for Telephony Services, 1-6
 - for Tserver operation, 1-6

D

- Data calls, 9-176
 - make call, 4-76
- Data structures, 9-1
- DCS
 - make call, 4-76
 - set do not disturb feature, 5-23
 - set forwarding feature, 5-27
- DCS network, event reporting, 9-177
- Deactivation
 - set forwarding feature, 5-28
- Deflect call
 - ack parameters, 4-61
 - description, 4-60
 - detailed information, 4-62
 - nak parameters, 4-61
 - overview, 4-5
 - service parameters, 4-61
 - syntax, 4-64
- Deflect from queue
 - deflect call, 4-62
 - pickup call, 4-115
- Delivered event
 - call coverage path to ACD device, 9-172
 - call scenarios, 9-51
 - deflect call, 4-62
 - description, 9-39
 - detailed information, 9-50
 - distributing device, 9-50
 - last redirection device, 9-50

- pickup call, 4-115
- private data v2-3 syntax, 9-70
- private data v4 syntax, 9-67
- private data v5 syntax, 9-63
- private data v6 syntax, 9-59
- private parameters, 9-45
- redirection, 9-165
- redirection on no answer, 9-165
- report, 9-39
- reports, consecutive, 9-40
- service parameters, 9-42
- syntax, 9-58
- userInfo parameter, 9-46, 9-48
- Designing applications, with screen pop information, 3-7
- Device
 - with bridged state, 7-8
- Device class, 3-29
- Device groups
 - trunk group, 3-29
- Device ID type
 - private data v2-4, 3-32
- Device identifier, 3-29
- Device identifiers
 - dynamic, 3-30
 - static, 3-29
 - syntax, 3-31
- Device monitoring event sequences
 - single step conference call, 4-153
- Device type, 3-28
 - ACD group, 3-28
 - definitions, 3-28
- Device type ID
 - private data v5-6, 3-32
- Device types
 - station, 3-28
 - trunks, 3-29
- Dialing, abbreviated, 9-131
- Digits collected
 - for screen pop, 3-7
- Direct agent calls
 - redirection on no answer, 9-165
- Direction
 - format, 3-45
- Display
 - make call, 4-76
 - make direct-agent call, 4-86
- Distributing device
 - delivered event, 9-50
- Diverted event
 - call coverage path (VDNs), 9-172
 - deflect call, 4-63
 - description, 9-73
 - detailed information, 9-75
 - pickup call, 4-115
 - redirection on no answer, 9-165
 - report, 9-73, 9-165
 - service parameters, 9-75
 - syntax, 9-76

Drop button
 single step conference call, 4-153

Drop button operation, 9-177
 clear connection, 4-22
 reconnect call, 4-22

DTMF receiver
 selective listening hold, 4-140
 selective listening retrieve, 4-140
 send DTMF tone, 4-133

DTMF sender
 send DTMF tone, 4-133

DTMF tones, unsupported, 4-133

Dynamic device identifier, 3-30

E

EnablePending private parameter, 5-10, 5-12

En-bloc sets, service initiated event, 9-178

Entered digits event
 description, 9-77
 detailed information, 9-77
 private parameter syntax, 9-79
 private parameters, 9-77
 service parameters, 9-77
 syntax, 9-78

Errors
 common CSTA, 3-49

Escape service group
 supported services, 3-5
 unsupported services, 3-6

Established event
 description, 9-80
 detailed information, 9-89
 private data v2-3 syntax, 9-105
 private data v4 syntax, 9-101
 private data v5 syntax, 9-97
 private data v6 syntax, 9-93
 private parameters, 9-85
 report, 9-80
 report, multiple, 9-81
 service parameters, 9-82
 syntax, 9-92
 userInfo parameter, 9-86, 9-88

Event filters, 8-3
 agent, 8-4
 call, 8-3
 feature, 8-4
 maintenance, 8-4

Event minimization feature, on G3 PBX, 9-2

Event report service group, 9-1
 supported services, 3-4
 unsupported services, 3-6

Event reports
 detailed information, 9-165
 monitor ended, 8-40

Events

advice of charge, 8-31
 call cleared, 9-3
 charge advice, 9-8
 conferenced, 9-13
 connection cleared, 9-32
 delivered, 9-39
 diverted, 9-73
 entered digits, 9-77
 established, 9-80
 failed, 9-109
 held, 9-114
 logged off, 9-116
 logged on, 9-119
 monitor ended, 8-2
 network reached, 9-122
 originated, 9-129
 queued, 9-135
 retrieved, 9-139
 route end, 10-2
 route register abort, 10-12
 route used (TSAPI v1), 10-54
 route used (TSAPI v2), 10-51
 service initiated, 9-142
 system status, 11-29
 system status, overview, 11-2
 transferred, 9-146

Expert Agent Selection (EAS), 9-177

F

Failed event
 description, 9-109
 detailed information, 9-112
 report, 9-109
 service parameters, 9-111
 syntax, 9-113

Feature access monitoring
 monitor device, 8-34

Feature availability
 charge advice event, 9-10
 single step conference call, 4-154

Feature event filters, 8-4

Feature summary
 for private data, 3-15

Feedback, interactions with call vectoring, 9-173

Filters
 agent event, 8-4
 call event, 8-3
 event feature, 8-4
 maintenance event, 8-4
 private, 8-4

Forced entry of account codes
 make call, 4-76

Format
 for G3 CSTA services, 3-45

Formats, 3-45

- ack parameters, 3-45
- ack private parameters, 3-45
- confirmation event, 3-45
- direction, 3-45
- function, 3-45
- nak parameters, 3-45
- private data, 3-45
- private parameters, 3-45
- service parameters, 3-45

Forwarded calls

- deflect call, 4-63
- pickup call, 4-116

Functional description

- conventions, 3-45

functional description, 3-45

G

- G3 CSTA system capacity, 3-40
- G3 local call state, mapped to CSTA local call state, 7-8
- G3 PBX
 - event minimization feature, 9-2

H

Held event

- description, 9-114
- detailed information, 9-114
- report, 9-114
- report, generating, 9-178
- service parameters, 9-114
- switch hook operation, 9-166
- syntax, 9-115

Held state

- alternate call, 4-67
- consultation call, 4-67
- hold call, 4-67

Hold button, 9-168

Hold call

- ack parameters, 4-66
- description, 4-65
- detailed information, 4-67
- nak parameters, 4-66
- overview, 4-5
- reconnect call, 4-128
- selective listening hold, 4-140
- selective listening retrieve, 4-140
- service parameters, 4-66
- syntax, 4-68

Hold state

- retrieve call, 4-128

Holding calls, generating held event report, 9-178

Hot line

- make call, 4-76

I

- Integrated Services Digital Network (ISDN), 9-178
- Interactions, between feedback and call vectoring, 9-173
- Interface structures
 - private data v4 syntax, 3-15
 - private data v5-6 syntax, 3-14
- Interflow, 9-169
- ISDN BRI station, single step conference call, 4-148

L

Last added party

- single step conference, 4-153

Last number dialed

- make call, 4-76

Last redirection device

- delivered event, 9-50
- established event, 9-90
- queued event, 9-137

Links

- multiple, considerations for, 3-43

Local call states, 7-8

LocalConnectionInfo parameter, for monitor services, 8-5

LocalConnectionState, definitions, 9-1

Logged off event

- description, 9-116
- detailed information, 9-116
- private parameter syntax, 9-118
- private parameters, 9-116
- report, 9-116
- service parameters, 9-116
- syntax, 9-117

Logged on event

- description, 9-119
- detailed information, 9-119
- private parameter syntax, 9-121
- private parameters, 9-119
- report, 9-119
- service parameters, 9-119
- syntax, 9-120

Logical agents, 9-177

- make call, 4-76
- make direct agent call, 4-86
- monitor device, 8-34
- set do not disturb feature, 5-23
- set forwarding feature, 5-27

Lookahead interflow, 9-175

Lookahead interflow info

- for screen pop, 3-7

Loop back

- deflect call, 4-63
- pickup call, 4-115

M

- Maintenance event filters, 8-4
- Maintenance service group
 - supported services, 3-5
 - unsupported services, 3-6
- Make call
 - ack parameters, 4-73
 - ack private parameters, 4-73
 - detailed information, 4-74
 - nak parameters, 4-73
 - overview, 4-6
 - private data v2-5 syntax, 4-81
 - private data v6 syntax, 4-79
 - private parameters, 4-72
 - service parameters, 4-71
 - syntax, 4-78
 - userInfo parameter, 4-72
- Make call service
 - description, 4-69
- Make direct-agent call
 - ack parameters, 4-84
 - ack private parameters, 4-85
 - description, 4-82
 - detailed information, 4-86
 - nak parameters, 4-85
 - overview, 4-6
 - private data v2-5 syntax, 4-90
 - private data v6 syntax, 4-88
 - private parameters, 4-84
 - service parameters, 4-83
 - syntax, 4-87
 - userInfo parameter, 4-94
- Make predictive call
 - ack parameters, 4-95
 - ack private parameters, 4-95
 - description, 4-91
 - detailed information, 4-96
 - nak parameters, 4-95
 - overview, 4-7
 - private data v2-5 syntax, 4-101
 - private data v6 syntax, 4-99
 - private parameters, 4-92
 - service parameters, 4-92
 - syntax, 4-98
- Make supervisor-assist call
 - ack parameters, 4-105
 - ack private parameters, 4-106
 - description, 4-103
 - detailed information, 4-107
 - nak parameters, 4-106
 - overview, 4-7
 - private data v2-5 syntax, 4-111
 - private data v6 syntax, 4-109
 - private parameters, 4-105
 - service parameters, 4-104
 - syntax, 4-108
 - userInfo parameter, 4-105
- Mandatory attributes, 3-45
- Manual transfer
 - established event, 9-90
- Maximum number of objects to monitor
 - monitor calls via device, 8-25
- Maximum requests from multiple G3PDs
 - monitor call, 8-16
- Monitor call
 - ack parameters, 8-15
 - ack private parameters, 8-15
 - description, 8-1, 8-13
 - detailed information, 8-16
 - nak parameters, 8-16
 - private data v2-4 syntax, 8-21
 - private data v5 syntax, 8-20
 - private parameters, 8-14
 - service parameters, 8-14
 - syntax, 8-18
- Monitor calls via device
 - ack parameters, 8-24
 - ack private parameters, 8-24
 - description, 8-2, 8-22
 - detailed information, 8-25
 - nak parameters, 8-25
 - private data v2-4 syntax, 8-30
 - private data v5 syntax, 8-29
 - private parameters, 8-23
 - service parameters, 8-23
 - syntax, 8-27
- Monitor device
 - ack parameters, 8-33
 - ack private parameters, 8-33
 - description, 8-2, 8-31
 - detailed information, 8-34
 - nak parameters, 8-34
 - private data v2-4 syntax, 8-39
 - private data v5 syntax, 8-38
 - private parameters, 8-33
 - service parameters, 8-32
 - syntax, 8-36
- Monitor ended event, 8-2
- Monitor ended event report, 8-40
 - detailed information, 8-40
 - monitor call, 8-16
 - service parameters, 8-40
 - syntax, 8-41
- Monitor service group
 - overview, 8-1
 - supported services, 3-4
- Monitor services, 8-3
 - localConnectionInfo parameter, 8-5
- Monitor stop
 - ack parameters, 8-46
 - description, 8-3, 8-46
 - detailed information, 8-46
 - nak parameters, 8-46
 - private parameters, 8-46

- syntax, 8-47
- Monitor stop on call, 8-42
 - ack parameters, 8-42
 - ack private parameters, 8-42
 - description, 8-2
 - detailed information, 8-43
 - nak parameters, 8-43
 - private parameters, 8-42
 - private parameters syntax, 8-45
 - syntax, 8-44
- Monitor stop on call service
 - monitor call, 8-16
- Multifunction
 - reconnect call, 4-14
- Multifunction station operation
 - alternate call, 4-14
 - answer call, 4-14
- Multiple application requests
 - monitor call, 8-17
- Multiple links
 - system status request, 11-5
- Multiple requests
 - monitor calls via device, 8-25
 - monitor device, 8-34
- Multiple split queueing, 9-175, 9-178
- Multiple telephony servers, 3-43
- Music on hold
 - alternate call, 4-67
 - consultation call, 4-67
 - hold call, 4-67
 - selective listening hold, 4-140
 - selective listening retrieve, 4-140
- MWI status sync
 - set MWI feature, 5-31

N

- Nak parameters
 - alternate call, 4-10
 - answer call, 4-14
 - change monitor filter, 8-8
 - change system status filter, 11-24
 - clear call, 4-18
 - clear connection, 4-22
 - conference call, 4-29
 - consultation call, 4-36
 - consultation direct-agent call, 4-45
 - consultation supervisor-assist call, 4-54
 - conventions, 3-45
 - deflect call, 4-61
 - hold call, 4-66
 - make call, 4-73
 - make direct-agent call, 4-85
 - make predictive call, 4-95
 - make supervisor-assist call, 4-106
 - monitor call, 8-16

- monitor calls via device, 8-25
- monitor device, 8-34
- monitor stop, 8-46
- monitor stop on call, 8-43
- pickup call, 4-114
- query ACD split, 6-3
- query agent login, 6-7
- query agent state, 6-15
- reconnect call, 4-121
- retrieve call, 4-127
- route end service (TSAPI v2), 10-8
- route register, 10-18
- route register cancel, 10-14
- route request (TSAPI v2), 10-24
- route select (TSAPI v2), 10-43
- selective listening hold, 4-139
- selective listening retrieve, 4-145
- set advice of charge, 5-3
- set agent state, 5-11
- set billing rate, 5-19
- set do not disturb feature, 5-23
- set forwarding feature, 5-27
- set MWI feature, 5-31
- single step conference call, 4-152
- system status request, 11-5
- system status start, 11-12
- system status stop, 11-19
- transfer call, 4-159

- Naming conventions
 - for structure members, 3-19
 - private data library, 3-19
- Naming conventions, for private data, 3-19
- Network reached event
 - description, 9-122
 - detailed information, 9-124
 - private data v2-4 syntax, 9-128
 - private data v5 syntax, 9-126
 - private parameters, 9-123
 - report, 9-122
 - service parameters, 9-123
 - syntax, 9-125
- Night service, 9-170
 - make call, 4-76

O

- Objects
 - connection, 3-35
 - device, 3-28
 - device type, 3-28
- Off-PBX destination
 - deflect call, 4-63
 - pickup call, 4-116
- Optional attributes, 3-45
- Original call info
 - to pop screen, 3-9

Originated event
 description, 9-129
 detailed information, 9-131
 private data v2-5 syntax, 9-134
 private data v6 syntax, 9-133
 private parameters, 9-130
 report, 9-129
 service parameters, 9-130
 syntax, 9-132
 userInfo parameter, 9-130

P

Parameters
 mandatory/optional attributes, 3-45
 Park/unpark call
 selective listening hold, 4-140
 selective listening retrieve, 4-140
 Party, last added
 single step conference call, 4-153
 PDU naming conventions, 3-19
 Personal Central Office Line (PCOL), 9-179
 make call, 4-76
 monitor calls via device, 8-25
 monitor device, 8-35
 Phantom calls, 3-28
 make call, 4-71
 make direct-agent call, 4-83
 make predictive call, 4-92
 make supervisor-assist call, 4-104
 Pickup call
 ack parameters, 4-114
 description, 4-113
 detailed information, 4-115
 nak parameters, 4-114
 overview, 4-7
 service parameters, 4-114
 syntax, 4-117
 PRI
 make call, 4-76
 Primary old call in conferenced event
 single step conference call, 4-154
 Primary Rate Interface (PRI), 9-179
 Priority calling
 make call, 4-77
 Priority calls
 deflect call, 4-63
 pickup call, 4-116
 Private data
 conventions, 3-19
 feature summary, 3-15
 version control, 3-12
 version feature support, 3-13
 Private data features
 initial DEFINITY release, 3-16
 initial G3PD release, 3-16

initial private data version, 3-16
 list of, 3-16
 Private data function
 convention, 3-45
 format, 3-45
 Private data interface structures
 v4 syntax, 3-15
 v5-6, 3-14
 Private data v5 to v6 migration, 3-19
 Private data v5, changes for private data v6, 3-19
 Private event parameters
 query agent login, 6-7
 Private filter, 8-4
 Private filter, set to on, 8-3
 Private parameters
 call cleared event, 9-4
 change monitor filter, 8-7
 change system status filter, 11-22
 charge advice event, 9-9
 clear connection, 4-21
 conferenced event, 9-16
 connection cleared event, 9-35
 consultation call, 4-35
 consultation direct-agent call, 4-44
 consultation supervisor-assist call, 4-53
 conventions, 3-45
 delivered event, 9-45
 entered digits event, 9-77
 established event, 9-85
 logged off event, 9-116
 logged on event, 9-119
 make call, 4-72
 make direct-agent call, 4-84
 make predictive call, 4-92
 make supervisor-assist call, 4-105
 monitor call, 8-14
 monitor calls via device, 8-23
 monitor device, 8-33
 monitor stop, 8-46
 monitor stop on call, 8-42
 network reached event, 9-123
 originated event, 9-130
 query ACD split, 6-3
 query agent login, 6-7
 query agent state, 6-14
 reconnect call, 4-120
 route request (TSAPI v2), 10-22
 route select (TSAPI v2), 10-40
 route used event (TSAPI v2), 10-52
 selective listening hold, 4-138
 selective listening retrieve, 4-144
 send DTMF tone, 4-131
 service initiated event, 9-143
 set advice of charge, 5-3
 set agent state, 5-9
 set billing rate, 5-19
 single step conference call, 4-149
 system status event, 11-31
 system status start, 11-11

transferred event, 9-149

Q

Query ACD split

- ack private parameters, 6-3
- description, 6-2
- nak parameters, 6-3
- private parameter syntax, 6-5
- private parameters, 6-3
- service parameters, 6-3
- syntax, 6-4

Query agent login

- ack parameters, 6-7
- ack private parameters, 6-7
- description, 6-6
- nak parameters, 6-7
- private event parameters, 6-7
- private parameter syntax, 6-11
- private parameters, 6-7
- service parameters, 6-7
- syntax, 6-9

Query agent state

- ack parameters, 6-14
- ack private parameters, 6-14
- description, 6-13
- detailed information, 6-15
- nak parameters, 6-15
- private data v2-4 syntax, 6-21
- private data v5 syntax, 6-20
- private data v6 syntax, 6-18
- private parameters, 6-14
- service parameters, 6-14
- syntax, 6-17

Query service group

- supported services, 3-3
- unsupported services, 3-5

Query trunk group

- overview, 6-1

Queued event

- description, 9-135
- detailed information, 9-136
- redirection on no answer, 9-165
- report, 9-135
- service parameters, 9-136
- syntax, 9-138

Queued event reports, multiple, 9-135

R

ReasonCode private parameter, 5-12

Reconnect call

- ack parameters, 4-120
- description, 4-118

- detailed information, 4-122

- nak parameters, 4-121

- overview, 4-8

- private data v2-5 syntax, 4-125

- private data v6 syntax, 4-124

- private parameters, 4-120

- service parameters, 4-119

- syntax, 4-123

- userInfo parameter, 4-120

Recording device, dropping

- single step conference call, 4-153

Release button, 9-168

Remote agent trunk

- single step conference call, 4-154

Remote applications, designing for, 3-10

Reports

- call cleared event, 9-3, 9-32

- charge advice event, 9-8

- conference event, 9-13

- connection cleared event, 9-32

- delivered event, 9-39

- delivered event, consecutive, 9-40

- diverted event, 9-73, 9-165

- established event, 9-80

- established event, multiple, 9-81

- failed event, 9-109

- held event, 9-114

- logged off event, 9-116

- logged on event, 9-119

- monitor ended event, 8-40

- network reached event, 9-122

- originated event, 9-129

- queued event, 9-135

- retrieved event, 9-139

- service initiated event, 9-142

- transferred event, 9-146

Requests, multiple

- monitor calls via device, 8-25

- monitor device, 8-34

Retrieve call, 4-27

- ack parameters, 4-127

- description, 4-126

- detailed information, 4-127

- nak parameters, 4-127

- overview, 4-8

- selective listening hold, 4-140

- selective listening retrieve, 4-140

- service parameters, 4-127

- syntax, 4-129

Retrieved event

- description, 9-139

- detailed information, 9-140

- report, 9-139

- service parameters, 9-140

- switch hook operation, 9-166

- syntax, 9-141

Ringback queueing, 9-180

Route end event

- description, 10-2
 - detailed information, 10-4
 - service parameters, 10-3
 - syntax, 10-5
 - Route end service (TSAPI v1)
 - description, 10-10
 - detailed information, 10-10
 - syntax, 10-11
 - Route end service (TSAPI v2)
 - ack parameters, 10-7
 - description, 10-6
 - detailed information, 10-8
 - nak parameters, 10-8
 - service parameters, 10-7
 - syntax, 10-9
 - Route register
 - ack parameters, 10-18
 - description, 10-17
 - detailed information, 10-18
 - nak parameters, 10-18
 - service parameters, 10-18
 - syntax, 10-19
 - Route register abort event
 - description, 10-12
 - detailed information, 10-12
 - service parameters, 10-12
 - syntax, 10-13
 - Route register cancel
 - ack parameters, 10-14
 - description, 10-14
 - detailed information, 10-15
 - nak parameters, 10-14
 - service parameters, 10-14
 - syntax, 10-16
 - Route request (TSAPI v1)
 - description, 10-36
 - detailed information, 10-36
 - syntax, 10-37
 - Route request (TSAPI v2)
 - ack parameters, 10-24
 - description, 10-20
 - detailed information, 10-24
 - nak parameters, 10-24
 - private data v2-4 syntax, 10-34
 - private data v5 syntax, 10-31
 - private data v6 syntax, 10-28
 - private parameters, 10-22
 - service parameters, 10-21
 - syntax, 10-26
 - Route select (TSAPI v1)
 - description, 10-49
 - detailed information, 10-49
 - syntax, 10-50
 - Route select (TSAPI v2)
 - ack parameters, 10-43
 - description, 10-39
 - detailed information, 10-43
 - nak parameters, 10-43
 - private data v2-5 syntax, 10-45, 10-47
 - private parameters, 10-40
 - service parameters, 10-40
 - syntax, 10-44
 - Route used event (TSAPI v1)
 - description, 10-54
 - detailed information, 10-54
 - private parameter syntax, 10-56
 - syntax, 10-55
 - Route used event (TSAPI v2)
 - description, 10-51
 - detailed information, 10-52
 - private parameters, 10-52
 - service parameters, 10-52
 - syntax, 10-53
 - Routing service group
 - supported services, 3-5
 - unsupported services, 3-6
 - Routing service group, overview, 10-1
-
- ## S
- Screen pop info
 - using original call info, 3-9
 - Screen pop information
 - called number, 3-7
 - calling number, 3-7
 - conferencing call, 3-7
 - digits collected by call prompting, 3-7
 - lookahead interflow information, 3-7
 - transferring call, 3-7
 - user-to-user information (UUI), 3-7
 - Security
 - single step conference call, 4-154
 - Selective listening hold
 - ack parameters, 4-138
 - description, 4-137
 - detailed information, 4-139
 - nak parameters, 4-139
 - private data v5 syntax, 4-142
 - private parameters, 4-138
 - service parameters, 4-138
 - syntax, 4-141
 - Selective listening retrieve
 - ack parameters, 4-144
 - description, 4-143
 - detailed information, 4-145
 - nak parameters, 4-145
 - private data v5 syntax, 4-147
 - private parameters, 4-144
 - service parameters, 4-144
 - syntax, 4-146
 - Send all call
 - pickup call, 4-115
 - Send All Calls (SAC), 9-180
 - make call, 4-77
 - set do not disturb feature, 5-24

- Send DTMF tone
 - ack parameters, 4-131, 4-132
 - description, 4-130
 - detailed information, 4-132
 - private data v4 syntax, 4-136
 - private data v5 syntax, 4-135
 - private parameters, 4-131
 - service parameters, 4-131
 - syntax, 4-134
- Send DTMF tone requests, multiple, 4-133
- Service availability
 - deflect call, 4-63
 - logged on event, 9-119
 - originated event, 9-131
 - pickup call, 4-116
 - send DTMF tone, 4-133
- Service groups
 - call control, 3-2
 - escape, 3-5
 - event report, 3-4
 - maintenance, 3-5
 - monitor, 3-4
 - query, 3-3
 - routing, 3-5
 - set feature, 3-3
 - snapshot, 3-3
 - supported, 3-1
 - system status, 3-5
- Service initiated event
 - description, 9-142
 - detailed information, 9-143
 - not sent with en-bloc sets, 9-178
 - private parameter syntax, 9-145
 - private parameters, 9-143
 - report, 9-142
 - service parameters, 9-143
 - switch hook operation, 9-166
 - syntax, 9-144
- Service observing, 9-170
- Service parameters, 10-12
 - alternate call, 4-10
 - answer call, 4-14
 - call cleared event, 9-4
 - change monitor filter, 8-7
 - change system status filter, 11-22
 - charge advice event, 9-9
 - clear call, 4-18
 - clear connection, 4-21
 - conference call, 4-28
 - conferenced event, 9-14
 - connection cleared event, 9-34
 - consultation call, 4-34
 - consultation direct-agent call, 4-43
 - consultation supervisor-assist call, 4-52
 - deflect call, 4-61
 - delivered event, 9-42
 - diverted event, 9-75
 - entered digits event, 9-77
 - established event, 9-82
 - failed event, 9-111
 - format, 3-45
 - held event, 9-114
 - hold call, 4-66
 - logged off event, 9-116
 - logged on event, 9-119
 - make call, 4-71
 - make direct-agent call, 4-83
 - make predictive call, 4-92
 - make supervisor-assist call, 4-104
 - monitor call, 8-14
 - monitor calls via device, 8-23
 - monitor device, 8-32
 - monitor ended event report, 8-40
 - network reached event, 9-123
 - originated event, 9-130
 - pickup call, 4-114
 - query ACD split, 6-3
 - query agent login, 6-7
 - query agent state, 6-14
 - queued event, 9-136
 - reconnect call, 4-119
 - retrieve call, 4-127
 - retrieved event, 9-140
 - route end event, 10-3
 - route end service (TSAPI v2), 10-7
 - route register, 10-18
 - route register cancel, 10-14
 - route request (TSAPI v2), 10-21
 - route select (TSAPI v2), 10-40
 - route used event (TSAPI v2), 10-52
 - selective listening hold, 4-138
 - selective listening retrieve, 4-144
 - send DTMF tone, 4-131
 - service initiated event, 9-143
 - set advice of charge, 5-3
 - set agent state, 5-7
 - set billing rate, 5-19
 - set do not disturb feature, 5-23
 - set forwarding feature, 5-27
 - set MWI feature, 5-31
 - single step conference call, 4-149
 - system status event, 11-30
 - system status request, 11-4
 - system status start, 11-11
 - system status stop, 11-19
 - transfer call, 4-158
 - transferred event, 9-147
- Service-observing, 9-180
- Services
 - alternate call, 4-9
 - alternate call, overview, 4-2
 - answer call, 4-13
 - answer call, overview, 4-2
 - change monitor filter, 8-1, 8-6
 - change system status filter, 11-21
 - change system status filter, overview, 11-2

- clear call, 4-18
- clear call, overview, 4-3
- clear connection, 4-20
- clear connection, overview, 4-3
- common errors, 3-49
- conference call, overview, 4-3
- consultation call, 4-33
- consultation call, overview, 4-4
- consultation direct-agent call, 4-42
- consultation direct-agent call, overview, 4-4
- consultation supervisor-assist call, 4-51
- consultation supervisor-assist call, overview, 4-5
- conventions for, 3-45
- deflect call, 4-60
- deflect call, overview, 4-5
- format for, 3-45
- hold call, 4-65
- hold call, overview, 4-5
- make call, 4-69
- make call, overview, 4-6
- make direct-agent call, 4-82
- make direct-agent call, overview, 4-6
- make predictive call, 4-91
- make predictive call, overview, 4-7
- make supervisor-assist call, 4-103
- make supervisor-assist call, overview, 4-7
- monitor, 8-3
- monitor call, 8-1, 8-13
- monitor call via device, 8-2, 8-22
- monitor device, 8-2, 8-31
- monitor stop, 8-3, 8-46
- monitor stop on call, 8-42
- monitor stop on call (private), 8-2
- pickup call, 4-113
- pickup call, overview, 4-7
- query ACD split, 6-2
- query agent login, 6-6
- query agent state, 6-13
- query trunk group, 6-1
- reconnect call, 4-118
- reconnect call, overview, 4-8
- retrieve call, 4-126
- retrieve call, overview, 4-8
- route end (TSAPI v1), 10-10
- route end (TSAPI v2), 10-6
- route register, 10-17
- route register cancel, 10-14
- route request (TSAPI v1), 10-36
- route request (TSAPI v2), 10-20
- route select (TSAPI v1), 10-49
- route select (TSAPI v2), 10-39
- selective listen retrieve, 4-143
- selective listening hold, 4-137
- send DTMF tone, 4-130
- set advice of charge, 5-2, 5-6, 8-31
- set billing rate, 5-18
- set do not disturb feature, 5-23
- set forwarding feature, 5-26
- set MWI feature, 5-31
- single step conference call, 4-148
- supported, 3-2
- supported groups, 3-1
- system status group, overview, 11-1
- system status request, 11-3
- system status request, overview, 11-1
- system status start, 11-10, 11-19
- system status stop, overview, 11-1
- transfer call, 4-157
- transfer call, overview, 4-8
- unsupported, 3-5
- Set advice of charge, 8-31
 - ack parameters, 5-3
 - ack private parameters, 5-3
 - description, 5-2
 - detailed information, 5-3
 - nak parameters, 5-3
 - private parameter syntax, 5-5
 - private parameters, 5-3
 - service parameters, 5-3
 - syntax, 5-4
- Set agent state
 - ack parameters, 5-11
 - ack private parameters, 5-11
 - description, 5-6
 - detailed information, 5-12
 - nak parameters, 5-11
 - private data v2-4 syntax, 5-17
 - private data v5 syntax, 5-16
 - private data v6 syntax, 5-15
 - private parameters, 5-9
 - service parameters, 5-7
 - syntax, 5-13
- Set billing rate
 - ack parameters, 5-19
 - description, 5-18
 - detailed information, 5-20
 - nak parameters, 5-19
 - private parameter syntax, 5-22
 - private parameters, 5-19
 - service parameters, 5-19
 - syntax, 5-21
- Set do not disturb feature
 - ack parameters, 5-23
 - description, 5-23
 - detailed information, 5-23
 - nak parameters, 5-23
 - service parameters, 5-23
 - syntax, 5-25
- Set feature service group
 - supported services, 3-3
- Set forwarding feature
 - ack parameters, 5-27
 - description, 5-26
 - nak parameters, 5-27
 - service parameters, 5-27
 - syntax, 5-29
- Set MWI feature
 - ack parameters, 5-31

- description, 5-31
- detailed information, 5-31
- nak parameters, 5-31
- service parameters, 5-31
- syntax, 5-33
- Single step conference call
 - ack parameters, 4-151
 - ack private parameters, 4-151
 - description, 4-148
 - detailed information, 4-153
 - feature availability, 4-154
 - nak parameters, 4-152
 - private data v5 syntax, 4-156
 - private parameters, 4-149
 - service parameters, 4-149
 - syntax, 4-155
- Single-digit dialing
 - make call, 4-77
- Skill hunt groups
 - make call, 4-77
 - monitor calls via device, 8-26
 - monitor device, 8-35
- Snapshot call service, 7-2
 - ack parameters, 7-3
 - CSTA connection states, 7-2
 - nak parameter, 7-3
 - service parameters, 7-3
 - syntax, 7-4
- Snapshot device service, 7-6
 - ack parameters, 7-7
 - ack private parameter, 7-7
 - nak parameter, 7-7
 - private data v2-4 syntax, 7-12
 - private data v5 syntax, 7-11
 - service parameters, 7-7
 - syntax, 7-9
- Snapshot service group
 - overview, 7-1
 - supported services, 3-3
- Split button, 9-168
- Start button, 9-168
- State of added station
 - single step conference call, 4-154
- Static device identifier, 3-29
- Station
 - monitor calls via device, 8-26
- Station device type, 3-28
- Station Message Detail Recording (SMDR)
 - make call, 4-77
- Structure members, naming conventions, 3-19
- Subdomain boundary, switching, 9-122
- Support for customers, 1-6
- Supported services, 3-1
- Switch administration
 - selective listening hold, 4-140
 - selective listening retrieve, 4-140
- Switch hook operation, 9-165
- Switch operation
 - after clear call, 4-18
 - alternate call, 4-67
 - clear connection, 4-22
 - consultation call, 4-67
 - hold call, 4-67
 - make call, 4-77
 - monitor stop, 8-46
 - reconnect call, 4-22, 4-128
 - retrieve call, 4-128
- Switch-hook flash field, 4-159
- Switching subdomain boundary, 9-122
- Synthesized message retrieval
 - set MWI feature, 5-32
- System capacity, 3-40
- System starts
 - set MWI feature, 5-32
- System status event
 - description, 11-29
 - detailed information, 11-31
 - overview, 11-2
 - private data v2-3 syntax, 11-35
 - private data v4 syntax, 11-34
 - private data v5 syntax, 11-33
 - private parameters, 11-31
 - service parameters, 11-30
 - syntax, 11-32
- System status events
 - not supported, 11-2
- System status group
 - unsupported services, 3-6
- System status request
 - ack parameters, 11-4
 - ack private parameters, 11-4
 - description, 11-3
 - detailed information, 11-5
 - multiple links, 11-5
 - nak parameters, 11-5
 - overview, 11-1
 - private data v2-3 syntax, 11-9
 - private data v4 syntax, 11-8
 - private data v5 syntax, 11-7
 - service parameters, 11-4
 - syntax, 11-6
- System status service group
 - supported services, 3-5
- System status service group, overview, 11-1
- System status start
 - ack parameters, 11-11
 - ack private parameters, 11-12
 - description, 11-10
 - detailed information, 11-13
 - nak parameters, 11-12
 - overview, 11-1
 - private data v2-3 syntax, 11-18
 - private data v4 syntax, 11-17
 - private data v5 syntax, 11-16
 - private parameters, 11-11
 - service parameters, 11-11

syntax, 11-14
 System status stop
 ack parameters, 11-19
 description, 11-19
 detailed information, 11-19
 nak parameters, 11-19
 overview, 11-1
 service parameters, 11-19
 syntax, 11-20

T

Talk state
 mapped to agent activity, 6-16
 Telephony servers
 multiple, 3-43
 Temporary bridged appearance
 clear connection, 4-23
 reconnect call, 4-23
 Temporary bridged appearances, 9-165, 9-180
 Terminating Extension Group (TEG), 9-181
 make call, 4-77
 monitor calls via device, 8-26
 monitor device, 8-35
 Tone cadence and level
 send DTMF tone, 4-133
 Transfer, 9-181
 Transfer call
 ack parameters, 4-158
 ack private parameters, 4-158
 description, 4-157
 detailed information, 4-159
 nak parameters, 4-159
 overview, 4-8
 private data v5 syntax, 4-162
 selective listening hold, 4-139
 selective listening retrieve, 4-139
 service parameters, 4-158
 syntax, 4-161
 Transferred event
 description, 9-146
 detailed information, 9-151
 private data v2-3 syntax, 9-162
 private data v4 syntax, 9-159
 private data v5 syntax, 9-156
 private data v6 syntax, 9-153
 private parameters, 9-149
 report, 9-146
 service parameters, 9-147
 syntax, 9-152
 trunkList parameter, 9-150
 userInfo parameter, 9-150
 Transferring call, with screen pop information, 3-7
 Transferring calls
 CSTA services used, 3-8
 Trunk group

 device type, 3-29
 Trunk group access, 9-169
 Trunk group administration
 charge advice event, 9-10
 Trunk to trunk transfer
 transfer call, 4-160
 TrunkList parameter
 conferenced event, 9-17
 transferred event, 9-150
 Trunks
 device types, 3-29
 Trunk-to-trunk transfer, 9-181
 TSAPI version control, 3-11

U

UserInfo parameter
 maximum size, 4-21, 4-35, 4-44, 4-53, 4-72, 4-94, 4-105,
 4-120, 9-17, 9-35, 9-46, 9-48, 9-86, 9-88, 9-150
 not supported by switch, 9-130
 User-to-user info
 passing info to remote applications, 3-10
 User-to-user information (UUI)
 for screen pop, 3-7

V

VDN
 make call, 4-74, 4-77
 monitor device, 8-35
 VDN destination
 make call, 4-77
 Vector-controlled split
 monitor calls via device, 8-26
 monitor device, 8-35
 Version control
 private data, 3-12
 TSAPI, 3-11
 Voice (synthesized) message retrieval
 set MWI feature, 5-32

W

WorkMode private parameter, 5-12

We'd like your opinion.

Avaya welcomes your feedback on this document. Your comments can be of great value in helping us improve our documentation.

CentreVu® Computer-Telephony®
Release 10.1, Version 1
Programmer's Guide for
DEFINITY® Enterprise Communications Server
Issue 1 — December 2001

1. Please rate the effectiveness of this document in the following areas:

	Excellent	Good	Fair	Poor
Ease of Finding Information				
Clarity				
Completeness				
Accuracy				
Organization				
Appearance				
Examples				
Illustrations				
Overall Satisfaction				

2. Please check the ways you feel we could improve this document:

- | | |
|---|---|
| <input type="checkbox"/> Improve the overview or introduction | <input type="checkbox"/> Make it more concise |
| <input type="checkbox"/> Improve the table of contents | <input type="checkbox"/> Add more step-by-step procedures/tutorials |
| <input type="checkbox"/> Improve the organization | <input type="checkbox"/> Add more troubleshooting information |
| <input type="checkbox"/> Add more figures | <input type="checkbox"/> Make it less technical |
| <input type="checkbox"/> Add more examples | <input type="checkbox"/> Add more or better quick reference aids |
| <input type="checkbox"/> Add more details | <input type="checkbox"/> Improve the index |

3. Please add details about your major concerns. _____

4. What did you like most about this document? _____

5. What did you like least about this document? _____

6. Feel free to write any comments below or on an attached sheet. _____

If we may contact you concerning your comments, please complete the following:

Name: _____ Telephone Number: (____) _____

Company/Organization: _____ Date: _____

Address: _____

Please FAX your response to (732) 817-5305.