Chapter 5SWITCHING FUNCTION SERVICESXE "SWITCHING FUNCTION SERVICES"§

This section describes Telephony Services which operate on calls and activate switch related features that are associated with the user desktop telephone or any other device defined by the switching domain. Switching Functions Services are divided into Basic Call Control Services and Telephony Supplementary Services.

Basic Call Control ServicesXE "Basic Call Control Services"§

This section defines Telephony Services which deal with basic call control for the desktop or call center environments. These functions provide services which allow client applications to:

- establish, control, and "tear-down" calls at a device or within the switch,
- answer incoming calls into a device, and
- activate/de-activate features and capabilities supported by the switch or the server.

Each function in this section has an associated confirmation event message which are events returned by the Telephony Server which indicate to the status and other function-specific information regarding the basic call control services request made by the application. Confirmation event messages are always returned as a result of a function call which has been successfully completed at the API Client Library. A confirmation event is always originated at the server once the application function has been processed by the server and/or the switch. If a call to a function is unsuccessful at the API Client Library level, the service request will not be sent to the Telephony Server and thus no confirmation event will be generated. If the function return code is anything other than success, the service request will not generate a confirmation event. The ***invokeIDXE " invokeID"§*** can be used to match a specific confirmation event with the specific function call which caused the event to be generated at the server.

Once an application receives a confirmation message to a service requested, e.g. receiving a **CSTAMakeCallConfEvent** after a **cstaMakeCall**( ) service request, the request has been processed by the server and the switch and the service request will either be successful or failed depending on the information which is returned in the confirmation event. The application should always check for a function confirmation event and possibly unsolicited status events (see *Status Reporting Services*) to ensure that a specific service request has been carried out by the server and/or the switch.

The application must have an active ACS Stream and an Event Handling Mechanism before confirmation events can be received from the Telephony Server. In addition, unsolicited status events also require an active monitor before status events are delivered to the application. See *Control Services* and *Status Reporting Services*, respectively, for more information on events.

Not every Driver implementation will support all Telephony functions. The application should use the cstaGetAPICaps function to determine which Telephony services are supported.

CSTAUniversalFailureConfEventXE "CSTAUniversalFailureConfEvent"§

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

Syntax

The following structure shows only the relevant portions of the unions for this message. See ***ACS Data Types*** and ***CSTA Data Types*** in section 4 for a complete description of the event structure.

```
        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 universalFailure;        } u;    }
cstaConfirmation;
    } event;} CSTAEvent_t;

typedef struct
{
    UniversalFailure_t    error;
} CSTAUniversalFailureConfEvent_t;


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,
    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 = 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
} CSTAUniversalFailure_t;
```

Parameters

acsHandle

This is the handle for the newly opened ACS Stream.

eventClass

This is a tag with the value **CSTACONFIRMATION**, which identifies this message as an CSTA confirmation event.

eventType

This tag with a value, **CSTA_UNIVERSAL_FAILURE_CONF**, identifies this message as an **CSTAUniversalFailureConfEvent.**

invokeID

This parameter specifies the function service request instance that has failed at the server or at the switch. This identifier is provided to the application when a service request is made.

error

Unspecified errorsXE "Unspecified errors"§

Error values in this category indicate that an error has occurred that is not among the other error types. This type includes the following specific error value:

Unspecified Error.

Operation errorsXE "Operation errors"§

Error values in this category indicate that there is an error in the Service Request. This type includes one of the following specific error values:

Generic Operation Error. This error indicate that the server has detected an error in the operation class, but that it is not one of the defined errors, or the server cannot be any more specific.

Request Incompatible With Object. The request is not compatible with the object.

Value Out Of Range. The parameter has a value that is not in the range defined for the server.

Object Not Known. The parameter has a value that is not known to the server.

Invalid Calling Device. The calling device is not valid.

Invalid Called Device. The called device is not valid.

Privilege Violation on Specified Device. The request cannot be provided because the specified device is not authorized for the Service.

Invalid Forwarding Destination. The request cannot be provided because the forwarding destination device is not valid.

Privilege Violation On Called Device. The request cannot be provided because the called device is not authorized for the Service.

Privilege Violation On Calling Device. The request cannot be provided because the calling device is not authorized for the Service.

Invalid CSTA Call Identifier. The call identifier is not valid.

Invalid CSTA Device Identifier. The Device Identifier is not valid.

Invalid CSTA Connection Identifier. The Connection identifier is not valid.

Invalid Destination. The Service Request specified a destination that is not valid.

Invalid Feature. The Service Request specified a feature that is not valid.

Invalid Allocation State. The Service Request indicated an allocation condition that is not valid.

Invalid Cross Reference ID The Service Request specified a Cross Reference Id that is not in use at this time.

Invalid Object Type. The Service Request specified an object type that is outside the range of valid object types for the Service.

Security Violation. The request violates a security requirement.

State incompatibility errors

XE "State incompatibility errors"§Error values in this category indicate that the Service Request was not compatible with the condition of a related CSTA object. This type includes the following specific error values:

Generic State Incompatibility. The server is unable to be any more specific.

Incorrect Object State. The object is in the incorrect state for the Service. This general error value may be used when the server isn't able to be any more specific.

Invalid CSTA Connection Identifier For Active Call. The Connection identifier specified in the Active Call parameter of the request is not in the correct state.

No Active Call. The requested Service operates on an active call, but there is no active call.

No Held Call. The requested Service operates on a held call, but the specified call is not in the Held state.

No Call To Clear. There is no call associated with the CSTA Connection identifier of the Clear Call request.

No Connection To Clear. There is no Connection for the CSTA Connection identifier specified as Connection To Be Cleared.

No Call To Answer. There is no call active for the CSTA Connection identifier specified as Call To Be Answered.

No Call To Complete. There is no call active for the CSTA Connection identifier specified as Call To Be Completed.

System resource availability errors

XE "System resource availability errors"§Error values in this category indicate that the Service Request cannot be completed because of a lack of system resources within the serving sub-domain. This type includes one of the following specific error values:

Generic System Resource Availability Error. The server is unable to be any more specific.

Service Busy. The Service is supported by the server, but is temporarily unavailable.

Resource Busy. An internal resource is busy. There is high probability that the Service will succeed if retried.

Resource Out Of Service. The Service requires a resource that is Out Of Service. A Service Request that encounters this condition could initiate system problem determination actions (e.g. notification of the network administrator).

Network Busy. The server sub-domain is busy.

Network Out Of Service. The server sub-domain is Out Of Service.

Overall Monitor Limit Exceeded. This request would exceed the server's overall limit of monitors.

Conference Member Limit Exceeded. This request would exceed the server's limit on the number of members of a conference.

Subscribed resource availability errors

XE "Subscribed resource availability errors"§Error values in this category indicate that the Service Request cannot be completed because a required resource must be purchased or contracted by the client system. This type includes the following specific error values:

Generic Subscribed Resource Availability Error. The server is unable to be any more specific.

Object Monitor Limit Exceeded. This request would exceed the server's limit of monitors for the specified object.

External Trunk Limit Exceeded. The limit of external trunks would be exceeded by this request.

Outstanding Requests Limit Exceeded. The limit of outstanding requests would be exceeded by this request.

Performance management errors

XE "Performance management errors"§Error values in this category indicate that an error has been returned as a performance management mechanism. This type includes the following specific error values:

Generic Performance Management Error. The server is unable to be any more specific.

Performance Limit Exceeded. A performance limit is exceeded.

7.*Security errorsXE "Security errors"§*

Error values in this category indicate that there is a security error. This type includes the following specific error values:

Generic Security Error. The server is unable to be any more specific.

Sequence Number Error. This error indicates that the server has detected an error in the Sequence Number of the operation.

Time Stamp Error. This error indicates that the server has detected an error in the Time Stamp of the operation.

PAC Error. This error indicates that the server has detected an error in the PAC of the operation.

Seal Error. This error indicates that the server has detected an error in the Seal of the operation.

privateData

If private data accompanied this event, then the private data would be copied to the location pointed to by the *privateData* pointer in the **acsGetEventBlock**( ) or **acsGetEventPoll**( ) function. If the *privateData* pointer is set to NULL in these functions, then no private data will be delivered to the application.

Comments

None.

cstaAlternateCall( )XE "cstaAlternateCall( )"§

The Alternate Call Service provides a higher-level, compound action of the Hold Call Service followed by Retrieve Call Service. This function will place an existing active call on hold and then either retrieves a previously held call or connects an alerting call at the same device.

Syntax

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

RetCode_t   cstaAlternateCall(
        ACSHandle_t             acsHandle,
        InvokeID_t              invokeID,
        ConnectionID_t          *activeCall,
        ConnectionID_t          *otherCall,
        PrivateData_t           *privateData);
```

Parameters

acsHandle

This is the value of the unique handle to the opened ACS Stream.

invokeID

A handle provided by the application to be used for matching a specific instance of a function service request with its associated confirmation event. This parameter is only used when the Invoke ID mechanism is set for Application-generated IDs in the **acsOpenStream**(**).** The parameter is ignored by the ACS Library when the Stream is set for Library-generated invoke IDs.

activeCall

This parameter points to the connection identifier for the "Connected" or active call which is to be alternated.

otherCall

This parameter points to the connection identifier for the "Alerting" or "Held" call which is to be alternated.

privateData

This is a pointer to the private data extension mechanism. Setting this parameter is optional. If the parameter is not used, the pointer should be set to NULL.

Return Values

This function returns the following values depending on whether the application is using library or application-generated invoke identifiers:

- *Library-generated Identifiers* - if the function call completes successfully it will return a positive value, i.e. the invoke identifier. If the call fails a negative error (<0) condition will be returned. For library-generated identifiers the return will never be zero (0).

- *Application-generated Identifiers* - if the function call completes successfully it will return a zero (0) value. If the call fails a negative error (<0) condition will be returned. For application-generated identifiers the return will never be positive (>0).

The application should always check the **CSTAAlternateCallConfEvent** message to ensure that the service request has been acknowledged and processed by the Telephony Server and the switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL

This return value indicates that a bad or unknown ***acsHandle*** was provided by the application.

ACSERR_STREAM_FAILED
This return value indicates that a previously active ACS Stream has been abnormally aborted.

CSTAERR_REQDENIED
This return value indicates that a ACS Stream is established but a requested capability has been denied by the Client Library Software Driver.

Comments

A successful call to this function will causes the held-or-delivered call to be swapped with the active call.

As shown in the figure below, the Alternate Call Service places the user's active call to device D2 on hold and, in a combined action, establishes or retrieves the call between device D1 and 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/active call to device D3.

The operation of the Alternate Call Service is depicted in Figure 5.1.

Before   After

Figure 5.1 - Alternate Call

CSTAAlternateCallConfEventXE "CSTAAlternateCallConfEvent"§

The Alternate Call confirmation event provides the positive response from the server for a previous alternate call request.

Syntax

The following structure shows only the relevant portions of the unions for this message. See ***ACS Data Types*** and ***CSTA Data Types*** in section 4 for a complete description of the event structure.

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

typedef struct
{
        ACSEventHeader_t    eventHeader;
        union
        {               struct
                {                       InvokeID_t        invokeID;
                        union
                        {
                                CSTAAlternateCallConfEvent_t    alternateCall;
                        } u;
                } cstaConfirmation;
        } event;
} CSTAEvent_t;

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

Parameters

acsHandle

This is the handle for the newly opened ACS Stream.

eventClass

This is a tag with the value **CSTACONFIRMATION**, which identifies this message as an CSTA confirmation event.

eventType

This is a tag with the value **CSTA_ALTERNATE_CALL_CONF**, which identifies this message as an **CSTAAlternateCallConfEvent.**

invokeID

This parameter specifies the function service request instance for the service which was processed at the Telephony Server or at the switch. This identifier is provided to the application when a service request is made.

privateData

If private data accompanied this event, then the private data would be copied to the location pointed to by the *privateData* pointer in the **acsGetEventBlock**( ) or **acsGetEventPoll**( ) function. If the *privateData* pointer is set to NULL in these functions, then no private data will be delivered to the application.

cstaAnswerCall( )XE "cstaAnswerCall( )"§

The Answer Call function will connect an alerting call at the device which is alerting. The call must be associated with a device that can answer a call without requiring physical user manipulation.

Syntax

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

RetCode_t   cstaAnswerCall (
        ACSHandle_t             acsHandle,
        InvokeID_t              invokeID,
        ConnectionID_t          *alertingCall,
        PrivateData_t           *privateData);
```

Parameters

acsHandle

This is the value of the unique handle to the opened ACS Stream.

invokeID

A handle provided by the application to be used for matching a specific instance of a function service request with its associated confirmation event. This parameter is only used when the Invoke ID mechanism is set for Application-generated IDs in the **acsOpenStream**( )**.** The parameter is ignored by the ACS Library when the Stream is set for Library-generated invoke IDs.

alertingCall

This parameter points to the connection identifier of the call to be answered.

privateData

This is a pointer to the private data extension mechanism. Setting this parameter is optional. If the parameter is not used, the pointer should be set to NULL.

Return Values

This function returns the following values depending on whether the application is using library or application-generated invoke identifiers:

- *Library-generated Identifiers* - if the function call completes successfully it will return a positive value, i.e. the invoke identifier. If the call fails a negative error (<0) condition will be returned. For library-generated identifiers the return will never be zero (0).

- *Application-generated Identifiers* - if the function call completes successfully it will return a zero (0) value. If the call fails a negative error (<0) condition will be returned. For application-generated identifiers the return will never be positive (>0).

The application should always check the **CSTAAnswerCallConfEvent** message to ensure that the service request has been acknowledged and processed by the Telephony Server and the switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
This return value indicates that a bad or unknown ***acsHandle*** was provided by the application.

ACSERR_STREAM_FAILED
This return value indicates that a previously active ACS Stream has been abnormally aborted.

CSTAERR_REQDENIED

This return value indicates that a ACS Stream is established but a requested capability has been denied by the Client Library Software Driver.

Comments

The Answer Call Service works for an incoming call that is alerting a device. In the following figure the call C1 is delivered to device D1. The **cstaAnswerCall**( ) is typically used with telephones that have attached speakerphone units to establish the call in a hands-free operation.

Before After

Figure 5.2 - Answer Call

CSTAAnswerCallConfEventXE "CSTAAnswerCallConfEvent"§

The Answer Call confirmation event provides the positive response from the server for a previous answer call request.

Syntax

The following structure shows only the relevant portions of the unions for this message. See ***ACS Data Types*** and ***CSTA Data Types*** in section 4 for a complete description of the event structure.

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

typedef struct
{
        ACSEventHeader_t    eventHeader;
        union
        {               struct
                {                       InvokeID_t         invokeID;
                        union
                        {
                                CSTAAnswerCallConfEvent_t    answerCall;
                        } u;
                } cstaConfirmation;
        } event;} CSTAEvent_t;

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

Parameters

acsHandle

This is the handle for the newly opened ACS Stream.

eventClass

This is a tag with the value **CSTACONFIRMATION**, which identifies this message as an CSTA confirmation event.

eventType

This is a tag with the value **CSTA_ANSWER_CALL_CONF**, which identifies this message as an **CSTAAnswerCallConfEvent.**

invokeID

This parameter specifies the function service request instance for the service which was processed at the Telephony Server or at the switch. This identifier is provided to the application when a service request is made.

privateData

If private data accompanied this event, then the private data would be copied to the location pointed to by the *privateData* pointer in the **acsGetEventBlock**( ) or **acsGetEventPoll**( ) function. If the *privateData* pointer is set to NULL in these functions, then no private data will be delivered to the application.

cstaCallCompletion( )XE " cstaCallCompletion( )"§

The Call Completion Service invokes specific switch features that may complete a call that would otherwise fail. The feature to be activated is passed as a parameter to the function.

Syntax

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

RetCode_t   cstaCallCompletion (
        ACSHandle_t             acsHandle,
        InvokeID_t              invokeID,
        Feature_t          feature,
        ConnectionID_t          *call,
        PrivateData_t           *privateData);
```

Parameters

acsHandle

This is the value of the unique handle to the opened ACS Stream.

invokeID

A handle provided by the application to be used for matching a specific instance of a function service request with its associated confirmation event. This parameter is only used when the Invoke ID mechanism is set for Application-generated IDs in the **acsOpenStream**(**).** The parameter is ignored by the ACS Library when the Stream is set for Library-generated invoke IDs.

feature

Specifies the call completion feature that is desired. These include:

CAMP_ON -	queues the call until the device is available.

CALL_BACK - requests the called device to return the call when it returns to idle.

INTRUDE - adds the caller to an existing active call at the called device. This feature requires the appropriate user security level at the server.

```
typedef enum Feature_t {
    FT_CAMP_ON = 0,
    FT_CALL_BACK = 1,
    FT_INTRUDE = 2
} Feature_t;
```

call

This is a pointer to a connection identifier for the call to be completed.

privateData

This is a pointer to the private data extension mechanism. Setting this parameter is optional. If the parameter is not used, the pointer should be set to NULL.

Return Values

This function returns the following values depending on whether the application is using library or application-generated invoke identifiers:

- *Library-generated Identifiers* - if the function call completes successfully it will return a positive value, i.e. the invoke identifier. If the call fails a negative error (<0) condition will be returned. For library-generated identifiers the return will never be zero (0).

- *Application-generated Identifiers* - if the function call completes successfully it will return a zero (0) value. If the call fails a negative error (<0) condition will be returned. For application-generated identifiers the return will never be positive (>0).

The application should always check the **CSTACallCompletionConfEvent** message to ensure that the service request has been acknowledged and processed by the Telephony Server and the switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
This return value indicates that a bad or unknown ***acsHandle*** was provided by the application.

ACSERR_STREAM_FAILED
This return value indicates that a previously active ACS Stream has been abnormally aborted.

CSTAERR_REQDENIED
This return value indicates that a ACS Stream is established but a requested capability has been denied by the Client Library Software Driver.

Comments

Generally this Service is invoked when a call is established and it encounters a busy or no answer at the far device.

The Camp On feature allows queuing for availability of the far end device. Generally, Camp On makes the caller wait until the called party finishes the current call and any previously camped on calls. Call Back allows requesting the called device to return the call when it returns to idle. Call Back works much like Camp On, but the caller is allowed to hang up after invoking the service, and the CSTA Switching Function calls both parties when the called party becomes free. Intrude allows the caller to be added into an existing call at the called device.

CSTACallCompletionConfEventXE "CSTACallCompletionConfEvent"§

The Call Completion confirmation event provides the positive response from the server for a previous call completion request.

Syntax

The following structure shows only the relevant portions of the unions for this message. See ***ACS Data Types*** and ***CSTA Data Types*** in section 4 for a complete description of the event structure.

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

typedef struct
{
        ACSEventHeader_t    eventHeader;
        union
        {               struct
                {                   InvokeID_t        invokeID;
                        union
                        {
                                CSTACallCompletionConfEvent_t         callCompletion;
                        }u;             } cstaConfirmation;
        } event;} CSTAEvent_t;

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

Parameters

acsHandle
This is the handle for the newly opened ACS Stream.

eventClass
This is a tag with the value **CSTACONFIRMATION**, which identifies this message as an CSTA confirmation event.

eventType

This is a tag with the value **CSTA_CALL_COMPLETION_CONF**, which identifies this message as an **CSTACallCompletionConfEvent.**

invokeID

This parameter specifies the function service request instance for the service which was processed at the Telephony Server or at the switch. This identifier is provided to the application when a service request is made.

privateData

If private data accompanied this event, then the private data would be copied to the location pointed to by the *privateData* pointer in the **acsGetEventBlock**( ) or **acsGetEventPoll**( ) function. If the *privateData* pointer is set to NULL in these functions, then no private data will be delivered to the application.

cstaClearCall( )XE "cstaClearCall( )"§

The Clear Call Service releases all of the devices from the specified call, and eliminates the call itself. The call ceases to exist and the connection identifiers used for observation and manipulation are released.

Syntax

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

RetCode_t   cstaClearCall (
        ACSHandle_t                     acsHandle,
        InvokeID_t                      invokeID,
        ConnectionID_t          *call,
        PrivateData_t           *privateData);
```

Parameters

acsHandle

This is the value of the unique handle to the opened ACS Stream.

invokeID

A handle provided by the application to be used for matching a specific instance of a function service request with its associated confirmation event. This parameter is only used when the Invoke ID mechanism is set for Application-generated IDs in the **acsOpenStream**(**).** The parameter is ignored by the ACS Library when the Stream is set for Library-generated invoke IDs.

call

This is a pointer to the connection identifier for the call to be cleared.

privateData

This is a pointer to the private data extension mechanism. Setting this parameter is optional. If the parameter is not used, the pointer should be set to NULL.

Return Values

This function returns the following values depending on whether the application is using library or application-generated invoke identifiers:

- *Library-generated Identifiers* - if the function call completes successfully it will return a positive value, i.e. the invoke identifier. If the call fails a negative error (<0) condition will be returned. For library-generated identifiers the return will never be zero (0).
- *Application-generated Identifiers* - if the function call completes successfully it will return a zero (0) value. If the call fails a negative error (<0) condition will be returned. For application-generated identifiers the return will never be positive (>0).

The application should always check the **CSTAClearCallConfEvent** message to ensure that the service request has been acknowledged and processed by the Telephony Server and the switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
This return value indicates that a bad or unknown ***acsHandle*** was provided by the application.

ACSERR_STREAM_FAILED
This return value indicates that a previously active ACS Stream has been abnormally aborted.

CSTAERR_REQDENIED
This return value indicates that a ACS Stream is established but a requested capability has been denied by the Client Library Software Driver.

Comments

This function will cause each device associated with a call to be released and the CSTA Connection Identifiers (and their components) are freed.

Figure 5.4 illustrates the results of a Clear Call (CSTA Connection ID = C1,D1), where call C1 connects devices D1, D2 and D3.

Before After

Figure 5.4 - Clear Call

CSTAClearCallConfEventXE "CSTAClearCallConfEvent"§

The Clear Call confirmation event provides the positive response from the server for a previous clear call request.

Syntax

The following structure shows only the relevant portions of the unions for this message. See ***ACS Data Types*** and ***CSTA Data Types*** in section 4 for a complete description of the event structure.

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

typedef struct
{
        ACSEventHeader_t    eventHeader;
        union
        {               struct
                {                   InvokeID_t        invokeID;
                        union
                        {
                                CSTAClearCallConfEvent_t  clearCall;
                        }u;             } cstaConfirmation;
        } event;} CSTAEvent_t;

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

Parameters

acsHandle

This is the handle for the newly opened ACS Stream.

eventClass

This is a tag with the value **CSTACONFIRMATION**, which identifies this message as an CSTA confirmation event.

eventType

This is a tag with the value

CSTA_CLEAR_CALL_CONF, which identifies this message as an **CSTAClearCallConfEvent.**

invokeID

This parameter specifies the function service request instance for the service which was processed at the Telephony Server or at the switch. This identifier is provided to the application when a service request is made.

privateData

If private data accompanied this event, then the private data would be copied to the location pointed to by the *privateData* pointer in the **acsGetEventBlock**( ) or **acsGetEventPoll**( ) function. If the *privateData* pointer is set to NULL in these functions, then no private data will be delivered to the application.

Comments

This confirmation indicates that all instances of the ACS Connection Identifiers for all the endpoints in the call and in the current association have become invalid. The instances of identifiers should not be used to request additional services of the Telephony Server.

cstaClearConnection( )XE "cstaClearConnection( )"§

The Clear Connection Service releases the specified device from the designated call. The Connection is left in the Null state. Additionally, the CSTA Connection Identifier provided in the Service Request is released.

Syntax

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

RetCode_t   cstaClearConnection (
        ACSHandle_t                    acsHandle,
        InvokeID_t                     invokeID,
        ConnectionID_t          *call,
        PrivateData_t           *privateData);
```

Parameters

acsHandle

This is the value of the unique handle to the opened ACS Stream.

invokeID

A handle provided by the application to be used for matching a specific instance of a function service request with its associated confirmation event. This parameter is only used when the Invoke ID mechanism is set for Application-generated IDs in the **acsOpenStream**( )**.** The parameter is ignored by the ACS Library when the Stream is set for Library-generated invoke IDs.

call

This is a pointer to the connection identifier for the connection to be cleared.

privateData

This is a pointer to the private data extension mechanism. Setting this parameter is optional. If the parameter is not used, the pointer should be set to NULL.

acsHandle

This is the value of the unique handle to the opened ACS Stream.

invokeID

A handle which can be provided by the application to match a specific instance of a function service request with its associated confirmation event. If the application provides an ***invokeID*** of zero (0), the API Client Library will select a unique positive invoke identifier on behalf of the application. A library-generated invoke identifier is returned upon a successful call to this function (***RetCode_t***). The invoke identifier can also be specified by the application. For application-generated invoke identifiers the invokeID parameter must be set to any non-zero value. In this case the API Client Library will not select an invoke identifier and the return value (***RetCode_t***) will return either zero (0) if successful or a negative error condition. In either case (library or application invoke identifiers), the ***invokeID*** for a specific service request will be included in its associated confirmation event.

Library-generated invoke identifiers will be created sequentially without regards to application-generated invoke identifiers. Mixing the two methods is not recommended since invoke identifiers should be unique.

Return Values

This function returns the following values depending on whether the application is using library or application-generated invoke identifiers:

- *Library-generated Identifiers* - if the function call completes successfully it will return a positive value, i.e. the invoke identifier. If the call fails a negative error (<0) condition will be returned. For library-generated identifiers the return will never be zero (0).

- *Application-generated Identifiers* - if the function call completes successfully it will return a zero (0) value. If the call fails a negative error (<0) condition will be returned. For application-generated identifiers the return will never be positive (>0).

The application should always check the **CSTAClearConnectionConfEvent** message to ensure that the service request has been acknowledged and processed by the Telephony Server and the switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
This return value indicates that a bad or unknown ***acsHandle*** was provided by the application.

ACSERR_STREAM_FAILED
This return value indicates that a previously active ACS Stream has been abnormally aborted.

CSTAERR_REQDENIED
This return value indicates that a ACS Stream is established but a requested capability has been denied by the Client Library Software Driver.

Comments

This Service releases the specified Connection and CSTA Connection Identifier instance from the designated call. The result is as if the device had hung up on the call. It is interesting to note that the phone may not be physically returned to the switch hook, which may result in silence, dial tone, or some other condition. Generally, if only two Connections are in the call, the effect of **cstaClearConnection**( ) function is the same as **cstaClearCall**( ).

Figure 5.5 is an example of the results of a Clear Connection (CSTA Connection Id = C1,D3), where call C1 connects devices D1, D2 and D3. Note that it is likely that the call is not cleared by this Service if it is some type of conference.

Before After

Figure 5.5 - Clear Connection

CSTAClearConnectionConfEventXE "CSTAClearConnectionConfEvent"§

The Clear Connection confirmation event provides the positive response from the server for a previous clear connection request.

Syntax

The following structure shows only the relevant portions of the unions for this message. See ***ACS Data Types*** and***CSTA Data Types*** in section 4 for a complete description of the event structure.

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

typedef struct
{
        ACSEventHeader_t    eventHeader;
        union
        {               struct
                {                   InvokeID_t          invokeID;
                                union
                                {
                                  CSTAClearConnectionConfEvent_t  clearConnection;
                                } u;
                } cstaConfirmation;
        } event;} CSTAEvent_t;

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

Parameters

acsHandle

This is the handle for the newly opened ACS Stream.

eventClass

This is a tag with the value **CSTACONFIRMATION**, which identifies this message as an CSTA confirmation event.

eventType

This tag with the value **CSTA_CLEAR_CONNECTION_CONF** identifies this message as an **CSTAClearConnectionConfEvent.**

invokeID

This parameter specifies the function service request instance for the service which was processed at the Telephony Server or at the switch. This identifier is provided to the application when a service request is made.

privateData

If private data accompanied this event, then the private data would be copied to the location pointed to by the *privateData* pointer in the **acsGetEventBlock**( ) or **acsGetEventPoll**( ) function. If the *privateData* pointer is set to NULL in these functions, then no private data will be delivered to the application.

Comments

This confirmation event indicates that the instance of the ACS Connection Identifier for the cleared Connection is released. The identifier should not be used to request additional services of the Telephony Server.

cstaConferenceCall( )XE " cstaConferenceCall( )"§

This function provides the conference of an existing held call and another active call at a device. The two calls are merged into a single call and the two Connections at the conferenceing device are resolved into a single Connection in the Connected state. The pre-existing CSTA Connection Identifiers associated with the device creating the conference are released, and a new CSTA Connection Identifier for the resulting conferenced Connection is provided.

Syntax

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

RetCode_t   cstaConferenceCall (
        ACSHandle_t                 acsHandle,
        InvokeID_t                  invokeID,
        ConnectionID_t          *heldCall,
        ConnectionID_t          *activeCall,
        PrivateData_t           *privateData);
```

Parameters

acsHandle

This is the value of the unique handle to the opened ACS Stream.

invokeID

A handle provided by the application to be used for matching a specific instance of a function service request with its associated confirmation event. This parameter is only used when the Invoke ID mechanism is set for Application-generated IDs in the **acsOpenStream**(**).** The parameter is ignored by the ACS Library when the Stream is set for Library-generated invoke IDs.

heldCall

This is a pointer to the connection identifier for the call which is on hold and is to be conferenced with an active call.

activeCall

This is a pointer to the connection identifier for the call which is active or proceeding and is to be conferenced with the held call.

privateData

This is a pointer to the private data extension mechanism. Setting this parameter is optional. If the parameter is not used, the pointer should be set to NULL.

Return Values

This function returns the following values depending on whether the application is using library or application-generated invoke identifiers:

- *Library-generated Identifiers* - if the function call completes successfully it will return a positive value, i.e. the invoke identifier. If the call fails a negative error (<0) condition will be returned. For library-generated identifiers the return will never be zero (0).

- *Application-generated Identifiers* - if the function call completes successfully it will return a zero (0) value. If the call fails a negative error (<0) condition will be returned. For application-generated identifiers the return will never be positive (>0).

The application should always check the **CSTAConferenceCallConfEvent** message to ensure that the service request has been acknowledged and processed by the Telephony Server and the switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
This return value indicates that a bad or unknown ***acsHandle*** was provided by the application.

ACSERR_STREAM_FAILED
This return value indicates that a previously active ACS Stream has been abnormally aborted.

CSTAERR_REQDENIED
This return value indicates that a ACS Stream is established but a requested capability has been denied by the Client Library Software Driver.

Comments

Figure 5.6 is an example of the starting conditions for the **cstaConferenceCall**( ) function, which are: the call C1 from D1 to D2 is in the held state. A call C2 from D1 to D3 is in progress or active.

Before After

Figure 5.6 - Conference Call

D1, D2 and D3 are conferenced or joined together into a single call, C3. The value of the Connection identifier (D1,C3) may be that of one of the CSTA Connection Identifiers provided in the request (D1,C1 or D1,C2).

CSTAConferenceCallConfEventXE "CSTAConferenceCallConfEvent"§

The Conference Call confirmation event provides the positive response from the server for a previous conference call request.

Syntax

The following structure shows only the relevant portions of the unions for this message. See ***ACS Data Types*** and ***CSTA Data Types*** in section 4 for a complete description of the event structure.

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

typedef struct
{
        ACSEventHeader_t    eventHeader;
        union
        {               struct
                {                       InvokeID_t        invokeID;                  union
                        {                       CSTAConferenceCallConfEvent_t  conferenceCall;                  } u;           }
cstaConfirmation;
        } event;} CSTAEvent_t;

typedef struct Connection_t {
   ConnectionID_t  party;
   DeviceID_t      staticDevice;
} Connection_t;

typedef struct ConnectionList {
   int            count;
   Connection_t    *connection;
} ConnectionList;

typedef struct CSTAConferenceCallConfEvent_t {
   ConnectionID_t  newCall;
   ConnectionList  connList;
} CSTAConferenceCallConfEvent_t;
```

Parameters

acsHandle

This is the handle for the newly opened ACS Stream.

eventClass

This is a tag with the value **CSTACONFIRMATION**, which identifies this message as an CSTA confirmation event.

eventType

This is a tag with the value **CSTA_CONFERENCE_CALL_CONF**, which identifies this message as an **CSTAClearConnectionConfEvent.**

invokeID

This parameter specifies the function service request instance for the service which was processed at the Telephony Server or at the switch. This identifier is provided to the application when a service request is made.

newCall

This parameter specifies the resulting connection identifier for the calls which were conferenced at the Conferenceing device. This connection identifier replaces the two previous connection identifier at that device.

connList

Specifies the resulting number of known devices in the conference. This field contains a count ***(count)*** of the number of devices in the conference and a pointer ***(*connection)*** to an array of Connection_t structures which define each connection in the call.

Each Connection_t record contains the following:

Party - indicates the Connection ID of the party in the conference.

Device - provides the static reference for the party in the conference. This parameter may have a value that indicates the static identifier is not known.

privateData

If private data accompanied this event, then the private data would be copied to the location pointed to by the *privateData* pointer in the **acsGetEventBlock**( ) or **acsGetEventPoll**( ) function. If the *privateData* pointer is set to NULL in these functions, then no private data will be delivered to the application.

cstaConsultationCall( )XE "cstaConsultationCall( )"§

The **cstaConsultationCall**( ) function will provide the compound or combined action of the Hold Call service followed by Make Call service. This service places an existing active call at a device on hold and initiates a new call from the same device using a single function call.

Syntax

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

RetCode_t   cstaConsultationCall (
        ACSHandle_t                 acsHandle,
        InvokeID_t                  invokeID,
        ConnectionID_t          *activeCall,
        DeviceID_t                  *calledDevice,
        PrivateData_t           *privateData);
```

Parameters

acsHandle

This is the value of the unique handle to the opened ACS Stream.

invokeID

A handle provided by the application to be used for matching a specific instance of a function service request with its associated confirmation event. This parameter is only used when the Invoke ID mechanism is set for Application-generated IDs in the **acsOpenStream**(**).** The parameter is ignored by the ACS Library when the Stream is set for Library-generated invoke IDs.

activeCall

This is a pointer to the connection identifier for the active call which is to be placed on hold before the new call is established.

calledDevice

This is a pointer to the destination device address for the new call to be established.

privateData
This is a pointer to the private data extension mechanism. Setting this parameter is optional. If the parameter is not used, the pointer should be set to NULL.

Return Values

This function returns the following values depending on whether the application is using library or application-generated invoke identifiers:

- *Library-generated Identifiers* - if the function call completes successfully it will return a positive value, i.e. the invoke identifier. If the call fails a negative error (<0) condition will be returned. For library-generated identifiers the return will never be zero (0).
- *Application-generated Identifiers* - if the function call completes successfully it will return a zero (0) value. If the call fails a negative error (<0) condition will be returned. For application-generated identifiers the return will never be positive (>0).

The application should always check the **CSTAConsultationCallConfEvent** message to ensure that the service request has been acknowledged and processed by the Telephony Server and the switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
This return value indicates that a bad or unknown ***acsHandle*** was provided by the application.

ACSERR_STREAM_FAILED
This return value indicates that a previously active ACS Stream has been abnormally aborted.

CSTAERR_REQDENIED
This return value indicates that a ACS Stream is established but a requested capability has been denied by the Client Library Software Driver.

Comments

This compound service allows the application to place an existing call on hold and at the same time establish a new call to another device.

In this case an active call C1 exists at D1 (see Figure 5.7) and a consultative call is desired to D3. After this function is called, the original active call (C1) is placed on hold and a new call, C2, is placed to device D3.

Before After

Figure 5.7 - Consultation Call

CSTAConsultationCallConfEventXE "CSTAConsultationCallConfEvent"§

The Consultation Call confirmation event provides the positive response from the server for a previous consultation call request.

Syntax

The following structure shows only the relevant portions of the unions for this message. See ***ACS Data Types*** and ***CSTA Data Types*** in section 4 for a complete description of the event structure.

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

typedef struct
{
            ACSEventHeader_t    eventHeader;
      union
      {           struct
            {                 InvokeID_t       invokeID;                union
                  {                 CSTAConsultationCallConfEvent_t consultationCall;
                  } u;          } cstaConfirmation;
      } event;} CSTAEvent_t;
      typedef struct CSTAConsultationCallConfEvent_t {
ConnectionID_t  newCall;
      } CSTAConsultationCallConfEvent_t;
```

Parameters

acsHandle
This is the handle for the newly opened ACS Stream.

eventClass
This is a tag with the value **CSTACONFIRMATION**, which identifies this message as an CSTA confirmation event.

eventType
This tag with the value **CSTA_CONSULTATION_CALL_CONF**, identifies

this message as an **CSTAConsultationCallConfEvent.**

invokeID
This parameter specifies the function service request instance for the service which was processed at the Telephony Server or at the switch. This identifier is provided to the application when a service request is made.

newCall
Specifies the Connection ID for the originating connection of the new call originated by the Consultation Call request.

privateData
If private data accompanied this event, then the private data would be copied to the location pointed to by the *privateData* pointer in the **acsGetEventBlock**( ) or **acsGetEventPoll**( ) function. If the *privateData* pointer is set to NULL in these functions, then no private data will be delivered to the application.

cstaDeflectCall( )XE " cstaDeflectCall( )"§

The **cstaDeflectCall**( ) service takes an alerting call at a device and redirects the call to another device on the switch.

Syntax

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

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

Parameters

acsHandle

This is the value of the unique handle to the opened ACS Stream.

invokeID

A handle provided by the application to be used for matching a specific instance of a function service request with its associated confirmation event. This parameter is only used when the Invoke ID mechanism is set for Application-generated IDs in the **acsOpenStream**(**).** The parameter is ignored by the ACS Library when the Stream is set for Library-generated invoke IDs.

deflectCall

This is a pointer to the connection identifier of the call which is to be deflected to another device within the switch.

calledDevice

A pointer to the device identifier where the original call is to be deflected.

privateData
This is a pointer to the private data extension mechanism. Setting this parameter is optional. If the parameter is not used, the pointer should be set to NULL.

Return Values

This function returns the following values depending on whether the application is using library or application-generated invoke identifiers:

- *Library-generated Identifiers* - if the function call completes successfully it will return a positive value, i.e. the invoke identifier. If the call fails a negative error (<0) condition will be returned. For library-generated identifiers the return will never be zero (0).

- *Application-generated Identifiers* - if the function call completes successfully it will return a zero (0) value. If the call fails a negative error (<0) condition will be returned. For application-generated identifiers the return will never be positive (>0).

The application should always check the **CSTADeflectCallConfEvent** message to ensure that the service request has been acknowledged and processed by the Telephony Server and the switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
This return value indicates that a bad or unknown ***acsHandle*** was provided by the application.

ACSERR_STREAM_FAILED
This return value indicates that a previously active

ACS Stream has been abnormally aborted.

CSTAERR_REQDENIED
This return value indicates that a ACS Stream is established but a requested capability has been denied by the Client Library Software Driver.

Comments

The Deflect Call Service takes a ringing (alerting) call at a device (D1) and sends it to a new destination (D3). This function replaces the original called device, as specified in the ***deflectCall*** parameter, with a different device within the switch, as specified in the ***calledDevice*** parameter.

Before After

Figure 5.8 - Deflect Call

CSTADeflectCallConfEventXE "CSTADeflectCallConfEvent"§

The Deflect Call confirmation event provides the positive response from the server for a previous deflect call request.

Syntax

The following structure shows only the relevant portions of the unions for this message. See ***ACS Data Types*** and ***CSTA Data Types*** in section 4 for a complete description of the event structure.

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

typedef struct
{
    ACSEventHeader_t   eventHeader;
    union
    {           struct
        {               InvokeID_t      invokeID;
                union
                {
                 CSTADeflectCallConfEvent_t  deflectCall;
                } u;
        } cstaConfirmation;
    } event;} CSTAEvent_t;

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

Parameters

acsHandle

This is the handle for the newly opened ACS Stream.

eventClass

This is a tag with the value **CSTACONFIRMATION**, which identifies this message as an CSTA confirmation event.

eventType

This is a tag with the value

CSTA_DEFLECT_CALL_CONF, which identifies this message as an **CSTADeflectCallConfEvent.**

invokeID

This parameter specifies the function service request instance for the service which was processed at the Telephony Server or at the switch. This identifier is provided to the application when a service request is made.

privateData

If private data accompanied this event, then the private data would be copied to the location pointed to by the *privateData* pointer in the **acsGetEventBlock**( ) or **acsGetEventPoll**( ) function. If the *privateData* pointer is set to NULL in these functions, then no private data will be delivered to the application.

cstaGroupPickupCall( )XE " cstaGroupPickupCall( )"§

The **cstaGroupPickupCall**( ) service moves an alerting call (at one or more devices in a device pickup group) to a specified device.

Syntax

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

RetCode_t   cstaGroupPickupCall (
        ACSHandle_t            acsHandle,
        InvokeID_t             invokeID,
        ConnectionID_t *deflectCall,
        DeviceID_t             *pickupDevice,
        PrivateData_t     *privateData);
```

Parameters

acsHandle
This is the value of the unique handle to the opened ACS Stream.

invokeID
A handle provided by the application to be used for matching a specific instance of a function service request with its associated confirmation event. This parameter is only used when the Invoke ID mechanism is set for Application-generated IDs in the **acsOpenStream**(**).** The parameter is ignored by the ACS Library when the Stream is set for Library-generated invoke IDs.

deflectCall
This is a pointer to the the call being picked up.

pickupDevice
This is a pointer to the device which is picking up calls from the group.

privateData

This is a pointer to the private data extension mechanism. Setting this parameter is optional. If the parameter is not used, the pointer should be set to NULL.

Return Values

This function returns the following values depending on whether the application is using library or application-generated invoke identifiers:

- *Library-generated Identifiers* - if the function call completes successfully it will return a positive value, i.e. the invoke identifier. If the call fails a negative error (<0) condition will be returned. For library-generated identifiers the return will never be zero (0).

- *Application-generated Identifiers* - if the function call completes successfully it will return a zero (0) value. If the call fails a negative error (<0) condition will be returned. For application-generated identifiers the return will never be positive (>0).

The application should always check the **CSTAGroupPickupConfEvent** message to ensure that the service request has been acknowledged and processed by the Telephony Server and the switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
This return value indicates that a bad or unknown ***acsHandle*** was provided by the application.

ACSERR_STREAM_FAILED
This return value indicates that a previously active ACS Stream has been abnormally aborted.

CSTAERR_REQDENIED
This return value indicates that a ACS Stream is established but a requested capability has been denied by the Client Library Software Driver.

Comments

The cstaGroupPickupCall( ) service redirects an alerting call (at one of more devices in a device pickup) to a specified device, the ***pickupDevice***.

Before After

Figure 5.10 - Group Pickup Call

CSTAGroupPickupCallConfEventXE "CSTAGroupPickupCallConfEvent"§

The Group Pickup Call confirmation event provides the positive response from the server for a previous Group Pickup call request.

Syntax

The following structure shows only the relevant portions of the unions for this message. See***ACS Data Types*** and ***CSTA Data Types*** in section 4 for a complete description of the event structure.

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

typedef struct
{
        ACSEventHeader_t    eventHeader;
        union
        {           struct
                {               InvokeID_t      invokeID;
                    union
                    {
                      CSTAGroupPickupCallConfEvent_t  groupPickupCall;
                    }u;         } cstaConfirmation;
        } event;} CSTAEvent_t;

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

Parameters

acsHandle

This is the handle for the newly opened ACS Stream.

eventClass

This is a tag with the value **CSTACONFIRMATION**, which identifies this message as an CSTA confirmation event.

eventType

This is a tag with the value **CSTA_GROUP_PICKUP_-CALL_CONF**, which identifies this message as an **CSTAGroupPickupCallConfEvent.**

invokeID

This parameter specifies the function service request instance for the service which was processed at the Telephony Server or at the switch. This identifier is provided to the application when a service request is made.

privateData

If private data accompanied this event, then the private data would be copied to the location pointed to by the *privateData* pointer in the **acsGetEventBlock**( ) or **acsGetEventPoll**( ) function. If the *privateData* pointer is set to NULL in these functions, then no private data will be delivered to the application.

cstaHoldCall( )XE "cstaHoldCall( )"§

The **cstaHoldCall**( ) service places an existing Connection in the held state.

Syntax

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

RetCode_t cstaHoldCall (
        ACSHandle_t                acsHandle,
        InvokeID_t                 invokeID,
        ConnectionID_t         *activeCall,
        Boolean_t              reservation,
        PrivateData_t          *privateData);
```

Parameters

acsHandle
This is the value of the unique handle to the opened ACS Stream.

invokeID
A handle provided by the application to be used for matching a specific instance of a function service request with its associated confirmation event. This parameter is only used when the Invoke ID mechanism is set for Application-generated IDs in the **acsOpenStream**( )**.** The parameter is ignored by the ACS Library when the Stream is set for Library-generated invoke IDs.

activeCall
A pointer to the connection identifier for the active call to be placed on hold.

reservation
Reserves the facility for reuse by the held call. This option is not appropriate for most non-ISDN telephones. The default is no connection reservation. This parameter is

optional.

privateData
This is a pointer to the private data extension mechanism. Setting this parameter is optional. If the parameter is not used, the pointer should be set to NULL.

Return Values

This function returns the following values depending on whether the application is using library or application-generated invoke identifiers:

- *Library-generated Identifiers* - if the function call completes successfully it will return a positive value, i.e. the invoke identifier. If the call fails a negative error (<0) condition will be returned. For library-generated identifiers the return will never be zero (0).

- *Application-generated Identifiers* - if the function call completes successfully it will return a zero (0) value. If the call fails a negative error (<0) condition will be returned. For application-generated identifiers the return will never be positive (>0).

The application should always check the **CSTAHoldCallConfEvent** message to ensure that the service request has been acknowledged and processed by the Telephony Server and the switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
This return value indicates that a bad or unknown ***acsHandle*** was provided by the application.

ACSERR_STREAM_FAILED
This return value indicates that a previously active ACS Stream has been abnormally aborted.

CSTAERR_REQDENIED
This return value indicates that a ACS Stream is established but a requested capability has been denied by the Client Library Software Driver.

Comments

A call to this function will interrupt communications for an existing call at a device. The call is usually, but not always, in the active state. A call may be placed on hold by the user some time after completion of dialing. The associated connection for the held call is made available for other uses, depending on the reservation option (ISDN-case). As shown in Figure 5.11, if the Hold Call service is invoked for device D1 on call C1, then call C1 is placed on hold at device D1. The hold relationship is affected at the holding device.

Before After

Figure 5.11 - Hold Call

The **cstaHoldCall**( ) service maintains a relationship between the holding device and the held call that lasts until the call is retrieved from the hold status, or until the call is cleared.