

*Chapter*

## **8** *CSTA COMPUTING FUNCTION SERVICES*

XE "CSTA:Computing Function Services"§XE "Computing Function Services"§CSTA Computing Functions are those functions where the switching domain is the client (service requester) and the computing domain is the server. Presently, Application Call Routing XE "Application Call Routing"§ is the only CSTA Computing Function. A switch uses application call routing when it needs the application to supply call destinations on a call-by-call basis. Applications can use internal databases together with call information to determine a destination for each call. For Example, an application might use the caller's number, caller-entered digits (provided as private data), or information in an application database to route incoming calls.

## Application Call Routing

XE "Application Call Routing"§ Application call routing requires that the switch be configured to direct calls to a special type of device known as the "routing device". When a call arrives at a routing device XE "Routing device"§, the switch sends a message to the Telephony Server requesting a route for the call.

XE "Routing Registration"§ Before an application can route calls, it must register with the Telephony Server as a routing server. The application may either register as the routing server for a specific routing device or as the default routing server XE "default routing server"§ for an advertised service XE "advertised service"§. Recall that a PBX driver XE "PBX driver"§ advertises its services. Often these services correspond to a CTI link XE "CTI link"§, so an application can, in effect, register to be the default routing server for a CTI link. An application uses **cstaRouteRegisterReq()** to register as a routing server. This request has an associated confirmation event, **CSTARouteRegisterReqConfEvent**.

At any one time, one, and only one application can be the routing server for a routing device. Similarly, one, and only one application can be the default routing server for an advertised service.

### Routing Procedure XE "Routing Procedure"§

1. **The switch queues an incoming call at a special device object, the routing device XE "Routing:Device"§. The routing device may be a "soft" extension on the switch for application-based routing, or similar device defined within the switching domain.**

2. **When the call arrives at the routing device, the switch and the Telephony Services PBX driver create a CSTA routing dialog for the call. An application uses a handle known as the *routing cross reference identifier* XE "Routing:Cross reference identifier" § (routingCrossRefID) to refer to this routing dialog.**
3. **The PBX Driver directs the route request to the application registered as the routing server for the routing device. If no application is registered for that specific routing device, then the PBX Driver directs the route request to the default routing server XE "default routing server" § application for its advertised service. The routing application receives an unsolicited *route request* XE "Route:Request " § event (CSTARouteRequestEvent) for the call. This event contains call related information (calling and called numbers). The application which provides the call routing destination is called the "routing server" for the routing device.**
4. **the routing server sends the switch a *route select* message (cstaRouteSelect) containing a destination for the call. The routing server typically uses information from the *route request* event together with information from an application database to determine this destination. The routing server may include an optional flag in the *route select* (*routeUsedReq*) instructing the switch to inform it of the final destination for the call. The final destination may be different than the application-provided destination when switch features such as call forwarding redirect the call.**
5. **the switch receives the *route select* message (cstaRouteSelect) and attempts to route the call to the**

application-provided destination. If the destination is valid, the switch routes the call to that destination and sends the application a *route end* event (`cstaRouteEnd`). This terminates the routing dialog for that call. If the application-provided destination is not valid (an `invalid` XE `"Route:Invalid"` § extension number, the destination is busy, etc.), then the switch may send a *re-route* XE `"Route:Re-route"` § event (`CSTAReRouteEvent`) to the application to request another route to a different destination.

6. If the application receives a *re-route* event (`CSTAReRouteEvent`) it can select a different destination for the call and send the switch another *route select* message (`cstaRouteSelect`). Depending on the switch implementation, the re-routing message exchange can repeat until the application provides a valid route. The routing server application will find out about a successful route XE `"Route:successful route"` § if the switch sends a *route end* event (`cstaRouteEnd`) or if the application included the *routeUsedReq* flag in its last *route select* message (`cstaRouteSelect`).

Either the switch or the routing server (application) may send a *route end* event (`cstaRouteEnd`) to end the routing process and terminate XE `"Routing:Terminating"` § the CSTA routing dialog (this invalidates the *routing cross reference identifier*, `routingCrossRefID`). Either endpoint may send a route end at any time. This message indicates that the routing server does not want to route the call, or the switch XE `"Routing:Switch default route"` § (usually in the absence of a `cstaRouteSelect` message) routed the call using some mechanism within the switching domain.

Certain switch implementations may not support the optional flags described above.

Figure 8-1 illustrates the Routing Procedure.

**Figure 8-2**

Routing Procedure  
Routing Procedure  
μ §

## **Routing Registration Functions and Events**

### **"Routing Registration Functions and Events"**

This section describes the service requests and events that an application uses to register with the Telephony Server as a call routing server

## **cstaRouteRegisterReq( )**XE "cstaRouteRegisterReq( )"§

An application uses **cstaRouteRegisterReq( )** to register as a routing server for a specific routing device or as a default routing server for an advertised service. The application must register for routing services before it can receive any route requests for a routing device. An application may be a routing server for more than one routing device. However, only one application may be a routing server for any given routing device. Similarly, only one application may register as the default routing server for an advertised service.

### **Syntax**

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

RetCode_t cstaRouteRegisterReq (
    ACSHandle_t      acsHandle,
    InvokeID_t       invokeID,
    DeviceID_t       *routingDevice,
    PrivateData_t    *privateData);
```

### **Parameters**

#### ***acsHandle***

This is the handle to an active ACS Stream over which the routing dialog will take place.

#### ***invokeID***

This is an application provided handle that the application uses to match a specific instance of **cstaRouteRegisterReq( )** request with its **CSTARouteRegisterReqConfEvent** confirmation event. The application supplies this parameter only when the Invoke ID mechanism is set for Application-generated IDs in **acsOpenStream( )**. The ACS Library ignores this parameter when the Stream is set for Library-generated invoke IDs.

### ***routingDevice***

This is a pointer to a device id for the routing device `XE "routing device"` for which the application requests to be the routing server. The routing device can be any device type which the switch implementation supports as a routing device. A NULL value indicates that the requesting application will be the default routing server `XE "default routing server"` for the **ServerID** associated with the **acsHandle** in the **cstaRouteRegisterReq()**. The default routing server will receive switch routing requests for any routing devices making routing requests on that advertised service that do not have registered routing servers. Thus, the default routing server will receive route requests when a routing device sends a route request and there is no corresponding registered routing server for that routing device.

### ***privateData***

This is an optional pointer to CSTA private data.

## **Return Values**

**cstaRouteRegisterReq()** 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, the invoke identifier. If the call fails it will return a negative error (<0). 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 it will return a negative error (<0). For application-generated identifiers the return is never positive (>0).

An application should always check the **CSTARouteRegisterReqConfEvent** message to ensure to ensure that the Telephony Server and switch have acknowledged the **cstaRouteRegisterReq( )**.

The following are possible negative error conditions for this function:

***ACSERR\_BADHDL***

The application provided a bad or unknown ***acsHandle***.

***ACSERR\_STREAM\_FAILED***

A previously active ACS Stream has been abnormally aborted.

**Comments**

In order for an application to route calls the application must successfully call **cstaRouteRegisterReq( )**. An application can register as:

- ◆ a routing server for the specified routing device, or
- ◆ as the default routing server for all routing devices making requests of a specific Telephony Server.

To register as a default routing server, an application sets the ***routingDevice*** parameter to NULL. One, and only one, application is allowed to register as the routing server for a ***routingDevice***, or as the default routing Server for an advertised service. Applications may register for routing services for a specific device even when a default routing server has registered. A default routing server will not receive any routing requests from any routing

device for which there is a registered routing server. Once a routing server is registered, **CSTARouteRequestEvents** convey the route requests to the routing server.

## CSTARouteRegisterReqConfEventXE "CSTARouteRegisterReqConfEvent"§

The **RouteRegisterReqConfEvent** indicates successful registration to an application. That application is now the call routing server for the requested routing device (or is the default routing server for the advertised service).

### 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
            {
                CSTARouteRegisterReqConfEvent_t routeReg;
            } u;
        } cstaConfirmation;
    } event;
} CSTAEvent_t;

typedef struct {
    RouteRegisterReqID_t routeRegisterReqID;
    CSTARouteRegisterReqConfEvent_t;
} typedef long RouteRegisterReqID_t;
```

### Parameters

#### ***acsHandle***

This is the handle for the ACS Stream over which the **RouteRegisterReqConfEvent** confirmation arrived. This is the same as the ACS Stream over which the application made the corresponding **cstaRouteRegisterReq( )** request.

***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\_ROUTE\_REGISTER**, which identifies this message as an **CSTARouteRegisterReqConfEvent**.

***invokeID***

This parameter specifies the service request instance for the **csaRouteRegisterReq()**. The application uses this parameter to correlate **RouteRegisterReqConfEvent** responses with requests.

***routeRegisterReqID***

This parameter contains a handle to the routing registration session for a specific routing device (or for the default routing server depending on the registration request). All routing dialogs (**routingCrossRefIDs**) for a routing device occur over this routing registration session. The PBX Driver selects **routeRegisterReqIDs** so that they will be unique within the **acsHandle**.

***privateData***

If private data accompanies this event, 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 **CSTARouteRegisterReqConfEvent** does not deliver private data to the application.

**Comments**

This event provides the application with a positive confirmation to a request for routing registration.



## **cstaRouteRegisterCancel( )**XE "cstaRouteRegisterCancel( )"§

Applications (routing servers) use **cstaRouteRegisterCancel( )** to cancel XE "Routing:Cancel"§ a previously registered routing server session. This request terminates the routing session and the application receives no further routing messages for that session once it receives the confirmation to the cancel request.

### **Syntax**

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

RetCode_t cstaRouteRegisterCancel (
    ACSHandle_t          acsHandle,
    InvokeID_t          invokeID,
    RouteRegisterReqID_t routeRegisterReqID,
    PrivateData_t       *privateData);
```

### **Parameters**

#### ***acsHandle***

This is the handle to an active ACS Stream over which the **cstaRouteRegisterCancel( )** request will be made.

#### ***invokeID***

This is an application provided handle that the application uses to match a specific instance of a **cstaRouteRegisterCancel( )** request with its confirmation event. The application supplies this parameter only when the Invoke ID mechanism is set for Application-generated IDs in **acsOpenStream( )**. The ACS Library ignores this parameter when the Stream is set for Library-generated invoke IDs.

#### ***routeRegisterReqID***

This parameter is the handle to the routing registration session which the application is canceling. The application received this handle in the confirmation event

for the route register service request, a **CSTARouteRegisterReqConfEvent**.

***privateData***

This is an optional pointer to CSTA private data.

**Return Values**

**cstaRouteRegisterCancel( )** 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, the invoke identifier. If the call fails it will return a negative error (<0). 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 it will return a negative error (<0). For application-generated identifiers the return is never positive (>0).

The application should always check the **CSTARouteRegisterCancelConfEvent** message to ensure that the Telephony Server and switch have acknowledged and processed the **cstaRouteRegisterCancel( )** request.

The following are possible negative error conditions for this function:

***ACSERR\_BADHDL***

The application provided a bad or unknown ***acsHandle***.

***ACSERR\_STREAM\_FAILED***

A previously active ACS Stream has been

abnormally aborted.

**Comments**

The application must continue to process outstanding routing requests from the routing device until it receives **CSTARouteRegisterCancelConfEvent**. The Telephony Server will not send any further requests once it has sent this confirmation event.

## CSTARouteRegisterCancelConfEventXE "CSTARouteRegisterCancelConfEvent"§

**CSTARouteRegisterCancelConfEvent** confirms a previously issued **cstaRouteRegisterCancel()** request for a routing registration. Once an application receives this event, it invalidates the routing registration session.

### 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
            {
                CSTARouteRegisterCancelConfEvent_t    routeCancel;    } u;
            } cstaConfirmation;
        } event; } CSTAEvent_t;

typedef struct {
    RouteRegisterReqID_t    routeRegisterReqID;}
CSTARouteRegisterCancelConfEvent_t;
typedef long    RouteRegisterReqID_t;
```

### Parameters

#### ***acsHandle***

This is the handle for the ACS Stream over which the **CSTARouteRegisterCancelConfEvent** confirmation arrived. This is the same as the ACS Stream over which the **cstaRouteRegisterCancel()** request was made.

***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\_ROUTE\_REGISTER\_CANCEL**, which identifies this message as an **CSTARouteRegisterCancelConfEvent**.

***invokeID***

This parameter specifies the service request instance for the **csaRouteRegisterCancel( )**. The application uses this parameter to correlate the **CSTARouteRegisterCancelConfEvent** responses with requests.

***routeRegisterReqID***

This parameter contains the handle to a routing registration for which the application is providing routing services. The application obtained this handle from a **CSTARouteRegisterReqConfEvent**. This **routeRegisterReqID** handle is no longer valid once the Telephony Server sends **CSTARouteRegisterCancelConfEvent**.

***privateData***

If private data accompanies this event, 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 **CSTASnapshotCallConfEvent** does not deliver private data to the application.

## **Comments**

**CSTARouteRegisterCancelConfEvent** confirms an application's **cstaRouteRegisterCancel()** service request, which cancels a routing registration session. The Telephony Server will send any further requests from the routing device to the default routing server.

## CSTARouteRegisterAbortEventXE "CSTARouteRegisterAbortEvent"§

The Telephony Server sends an application an unsolicited **CSTARouteRegisterAbortEvent** to cancel **XE** "Routing:Cancel"§ an active routing dialog. This event invalidates a routing registration session.

### 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
        {
            union
            {
                CSTARouteRegisterAbortEvent_t routeCancel;
            } u;      } cstaEventReport;
    } event;} CSTAEvent_t;
typedef struct {
    RouteRegisterReqID_t      routeRegisterReqID;}
CSTARouteRegisterAbortEvent_t;
typedef long    RouteRegisterReqID_t;
```

### Parameters

#### ***acsHandle***

This is the handle for the opened ACS Stream. The routing dialog being canceled is occurring on this ACS Stream.

#### ***eventClass***

This is a tag with the value **CSTAEVENTREPORT**,

which identifies this message as an CSTA event report.

***eventType***

This is a tag with the value **CSTA\_ROUTE\_REGISTER\_ABORT**, which identifies this message as an **CSTARouteRegisterAbortEvent**.

***routeRegisterReqID***

This parameter is the handle to a routing registration for which the application is providing routing services. The application received this handle in a **CSTARouteRegisterReqConfEvent**. The **CSTARouteRegisterAbortEvent** invalidates this handle.

***privateData***

If private data accompanies **CSTARouteRegisterAbortEvent**, 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 **CSTARouteRegisterAbortEvent** does not deliver private data to the application.

**Comments**

**CSTARouteRegisterAbortEvent** notifies the application that the PBX driver or switch aborted a routing registration session.

## Routing Functions and Events

### "Routing:Functions" § "Routing:Events" §

This section defines the CSTA call routing services for application call routing. The switch queues calls at the routing device `"Routing:Device" §` until the application provides a destination for the call or a time-out condition occurs within the switching domain. Figure 8-3 shows the Application-based call routing dialogue between a switch and the routing server (the application).

Once an application registers as a routing server, the application uses the services in this section to route calls. The application receives a **CSTARouteRequestEvent** for each call which requires a routing destination. The application sends the switch a destination in **cstaRouteSelect()**. The switch then attempts to route the call to that application-provided destination. The switch will respond with a **CSTARouteEndEvent** and/or a **CSTARouteUsedEvent** `"Route:Used" § t`. If the application-provided destination is invalid, the switch may send a **CSTAReRouteEvent** `"Route:Re-route" §` to request an additional destination. See Figure 8-4 for a typical sequence of these events and service requests.

## Register Request ID and the Routing Cross Reference ID

XE "Register Request ID"§XE "Routing Cross Reference ID"§The routing services use two handlesXE "Handle:Register Request ID"§XE "Handle:Routing Cross Reference ID"§ (identifiers) to refer to different software objects in the Telephony Server. The register request identifier (**routeRegisterReqID**) identifies a routing session over which an application will receive routing requests. This handle is tied to a routing deviceXE "Routing:Device"§ on the switch, or it may indicate that the application is the default routing serverXE "Routing:Default routing server"§XE "Default routing server:See Routing (Default Routing Server)"§ for an advertised service. When the application uses **cstaRouteRegisterReq()** to register for routing services, it receives a **routeRegisterReqID** in the confirmation. The **routeRegisterReqID** is valid until the registration is canceled or abortedXE "routeRegisterReqID:Duration"§.

Within a routing session (**routeRegisterReqID**) the switch may initiate many routing dialogs (shown in Figure 8-5) to route multiple calls. An application uses a routing cross reference identifier (**routingCrossRefID**) to refer to each routing dialog. The application receives a **routingCrossRefID** in each **CSTARouteRequestEvent**. The **CSTARouteRequestEvent** initiates a routing dialog. The **routingCrossRefID** is valid for the duration of the call routing dialogXE "routingCrossRefID:Duration"§.

The routing cross reference identifier (**routingCrossRefID**) is uniqueXE "routingCrossRefID:Uniqueness"§ within the routing session (**routeRegisterReqID**). Some switch implementationsXE "Switch:Specific"§ may provide the additional benefit of a unique routing cross reference identifier across the entire switching domain. Routing session identifiers

(**routeRegisterReqIDs**) are unique XE  
"routeRegisterReqID:Uniqueness"§ within an ACS Stream  
(**acsHandle**).

If a call is not successfully routed by the routing server this does not necessarily mean that the call is cleared or not answered. Most switch implementations will have a default mechanism for handling a call at a routing device when the routing server has failed to provide a valid destination for the call.

## **CSTARouteRequestEventXE "CSTARouteRequestEvent"\$XE "CSTARouteRequestEvent"\$**

A routing server application receives a **CSTARouteRequestEvent** when the switch requests a route for a call. 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 for the advertised service. The **CSTARouteRequestEvent** event includes call related information (such as the called and calling number, when available). A routing server application typically combines the call related information with an application database to determine a destination for the call. A routing server application receives a **CSTARouteRequestEvent** for every call queued at the routing device.

### **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  
            {  
                CSTARouteRequestEvent_t  routeRequest;  
            } u;  
        } cstaRequestEvent;  
    } event;} CSTAEvent_t;
```

```

typedef struct {
    RouteRegisterReqID_t      routeRegisterReqID;
    RoutingCrossRefID_t      routingCrossRefID;
    CalledDeviceID_t         currentRoute;
    DeviceID_t               currentRoute;
    CallingDeviceID_t        callingDevice;
    DeviceID_t               callingDevice;
    ConnectionID_t           routedCall;
    SelectValue_t            routedSelAlgorithm;
    Boolean                  priority;
    SetupValues_t            setupInformation;
} CSTARouteRequestEvent_t;

typedef enum SelectValue_t {
    SV_NORMAL = 0,
    SV_LEAST_COST = 1,
    SV_EMERGENCY = 2,
    SV_ACD = 3,
    SV_USER_DEFINED = 4
} SelectValue_t;
typedef struct SetupValues_t { int length; unsigned char *value;} SetupValues_t;

```

## Parameters

### *acsHandle*

This is the handle for the opened ACS Stream on which the route request event arrives.

### *eventClass*

This is a tag with the value **CSTAREQUEST**, which identifies this message as an CSTA request message.

### *eventType*

This is a tag with the value **CSTA\_ROUTE\_REQUEST**, which identifies this message as an **CSTARouteRequestEvent**.

### *monitorCrossRefID*

**Does not apply to this event.**

### *routeRegisterReqID*

This parameter 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

route register service request.

***routingCrossRefID***

The application receives this new handle for the routing dialog for this call. This identifier has a new, unique value within the scope of the routing session (***routeRegisterReqID***).

***currentRoute***

This parameter indicates the originally requested destination for the call being application routed. Often, this is the DNIS, or dialed number.

***callingDevice***

This is the originating device of the call, i.e. the calling party number (when available. If not available, it may be trunk information).

***routedCall***

This parameter is a CSTA Connection ID that identifies the call being routed.

***routedSelAlgorithm***

This parameter identifies the routing algorithm being used.

***priority***

This parameter indicates the priority of the call.

***setupInformation***

This parameter includes an ISDN call setup message, if available.

***privateData***

If private data accompanies **CSTARouteRequestEvent**, 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 **CSTARouteRequestEvent** does not deliver private data to the application.

### **Comments**

**CSTARouteRequestEvent** informs the routing server (application) that the switch is requesting a destination for a call queued at the routing device. The application uses **cstaRouteSelect( )** to respond with a destination.

CSTA requires that all events contain an invoke ID. During routing, the *RouteRegisterReqID* and the *RoutingCrossRefID* identify the routing dialogue. The *invokeID* is not used.

## CSTAReRouteEventXE "CSTAReRouteEvent"§

The switch sends an unsolicited **CSTAReRouteEvent** to request an another destination for a call. Typically, the destination that the application previously sent was invalid or busy. The switch previously sent Call related information (such as the called and calling numbers) in the **CSTARouteRequestEvent**; Call related information is not re-sent in the **CSTAReRouteEvent**. The routing server application responds using the **csaRouteSelect()** service.

### Syntax

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

typedef struct
{
    ACSEventHeader_t  eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID;
            union
            {
                CSTAReRouteRequest_t reRouteRequest;
            } u;
        } cstaRequestEvent;
    } event;} CSTAEvent_t;

typedef struct
{
    RouteRegisterReqID_t      routeRegisterReqID;
    RoutingCrossRefID_t      routingCrossRefID;
} CSTAReRouteEvent_t;
```

### Parameters

#### ***acsHandle***

This is the handle for the opened ACS Stream on which the re-route request arrives.

#### ***eventClass***

This is a tag with the value **CSTAREQUEST**, which

identifies this message as a CSTA request message.

***eventType***

This is a tag with the value **CSTA\_RE\_ROUTE\_REQUEST**, which identifies this message as a **CSTAReRouteEvent**.

***monitorCrossRefID***

***Does not apply to this event.***

***routeRegisterReqID***

This parameter 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 route register service request.

***routingCrossRefID***

This parameter contains the handle to the CSTA call routing dialog for this call. The application previously received this handle in a **CSTARouteRequestEvent** for the call.

***privateData***

If private data accompanies **CSTAReRouteEvent**, 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 **CSTAReRouteEvent** does not deliver private data to the application.

**Comments**

The switch can send **CSTAReRouteEvent** to the routing server application when the application previously sent a destination that was is invalid or other circumstances

exist where routing of the call to the destination is not possible (e.g. the destination is busy). The switch uses **CSTAReRouteEvent** to request another destination for the call queued at the routing device. The application uses **cstaRouteSelect( )** to provide the new destination.

The number of re-route requests that a switch XE "Switch:Specific"§ may send depends on the implementation or administration within the switch. The application should be prepared to respond to all re-route requests or terminate the routing dialog by using the **cstaRouteEnd( )** service request when it cannot provide additional destinations.

CSTA requires that all events contain an invoke ID. During routing, the RouteRegisterReqID and the RoutingCrossRefID identify the routing dialogue. The invokeID is not used.

## **cstaRouteSelect( )XE "cstaRouteSelect( )"S**

The routing server application uses **cstaRouteSelect** to send a routing destination to the switch in response to a **CSTARouteRequestEvent** for a call.

### **Syntax**

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

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);
```

### **Parameters**

#### ***acsHandle***

This is the handle to the ACS Stream on which the routing dialog for the call is taking place.

#### ***routeRegisterReqID***

This parameter contains the active handle to the routing registration session for which the application is providing routing services. The application received this handle in the confirmation event for the route register service request, **CSTARouteRegisterReqConfEvent**, for the call.

#### ***routingCrossRefID***

This parameter contains the handle to the CSTA call routing dialog for this call. The application previously received this handle in the **CSTARouteRequestEvent** for the call.

The application passes the routingCrossRefID for the call

that it previously received in the application received this handle in the confirmation event for the route register service request, **CSTARouteRegisterReqConfEvent**, for the call.

***routeSelected***

The application provides this parameter containing a Device ID that specifies the destination for the call.

***remainRetry***

The application indicates the number of times it is willing to receive a **CSTAReRouteRequestEvent** for this call in the case that the switch needs to request an alternate route. This element may have a special value that shall indicate that the routing server does not keep count, or that there is no limit.

***setupInformation***

The application provides this optional parameter that contains information for the ISDN call setup message that the switch will use to route the call. Some switches XE "Switch:Specific"§ may not support this option.

***routeUsedReq***

The routing application uses this parameter to request a **CSTARouteUsedEvent** for the call. The route used event informs the application of the final destination of the call once it has been routed.

***privateData***

This is an optional pointer to CSTA private data.

**Return Values**

**cstaRouteSelect( )** returns a non-zero value if it completes successfully.

The following are possible negative error conditions for this function:

***ACSERR\_BADHDL***

The application provided a bad or unknown ***acsHandle***.

***ACSERR\_STREAM\_FAILED***

A previously active ACS Stream has been abnormally aborted.

There is no confirmation event for this service request, however this service request can generate a universal failure event.

**Comments**

An application should call ***cstaRouteSelect*** only in response to a ***CSTARouteRequestEvent***. The ***cstaRouteSelect*** service request will fail if the application does not provide valid identifiers from a previous ***CSTARouteRequestEvent***, (***acsHandle***, ***routeRegisterReqID***, and ***routingCrossRefID***). The application should check the return value for this function and any resulting universal failure event for errors.

## CSTARouteUsedEventXE "CSTARouteUsedEvent"§

The **CSTARouteUsed** event provides a routing server application with the actual destinationXE "Routing:Actual destination"§ of a call for which the application previously sent a **cstaRouteSelect( )**. To receive a **CSTARouteUsed**, the application must set the **cstaRouteSelect( )** parameter **routeUsedReq** to TRUE when it sends the **cstaRouteSelect( )**.

### 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
        {
            union
            {
                CSTARouteUsedEvent_t routeUsed;
            } u;
        } cstaEventReport;
    } event;} CSTAEvent_t;

typedef struct
{
    RouteRegisterReqID_t      routeRegisterReqID;
    RoutingCrossRefID_t      routingCrossRefID;
    DeviceID_t                routeUsed;
    DeviceID_t                callingDevice;
    Boolean                    domain;
} CSTARouteUsedEvent_t;
```

### Parameters

#### **acsHandle**

This is the handle to the ACS Stream on which the

routing dialog for the call is taking place.

***eventClass***

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

***eventType***

This is a tag with the value **CSTA\_ROUTE\_USED**, which identifies this message as an **CSTARouteUsedEvent**.

***routeRegisterReqID***

This parameter contains the active handle to the routing registration session for which the application is providing routing services. The application received this handle in the confirmation event for the route register service request, **CSTARouteRegisterReqConfEvent**, for the call.

***routingCrossRefID***

This parameter contains the handle to the CSTA call routing dialog for this call. The application previously received this handle in the **CSTARouteRequestEvent** for the call. This routing cross reference ID for the call will match a routing cross reference ID that the application previously received in the **CSTARouteRegisterReqConfEvent** for the call.

***routeUsed***

This parameter identifies the selected and final destination for the call.

***callingDevice***

This parameter contains the originating device of the call, i.e. the calling party number (when available).

***domain***

This parameter will indicate whether the call has left the switching domain accessible to the Telephony Server (the ***ServerID*** defined in the active ***acsHandle***). Typically, a call leaves a switching domain when it is routed to a trunk connected to another switch or to the public switched network.

***privateData***

If private data accompanies **CSTARouteUsedEvent**, 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 **CSTARouteUsedEvent** does not deliver private data to the application.

**Comments**

An application uses **CSTARouteUsedEvent** to determine the final destination of a call that it routed using the **cstaRouteSelect( )**. Switch features such as forwarding or routing tables may direct the call to a device other than the application supplied destination. The **CSTARouteUsedEvent** indicates the final destination for the call.

## CSTARouteEndEventXE "CSTARouteEndEvent"§

The switch sends **CSTARouteEndEvent** to terminate a routing dialog. The event includes a cause value giving the reason for the dialog termination.

### 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
        {
            union
            {
                CSTARouteEndEvent_t routeEnd,
            } u;
        } cstaEventReport;
    } event; } CSTAEvent_t;

typedef struct
{
    RouteRegisterReqID_t  routeRegisterReqID;
    RoutingCrossRefID_t routingCrossRefID;
    CSTAUniversalFailure_t  errorValue;
} CSTARouteEndEvent_t;
```

### Parameters

#### *acsHandle*

This is the handle for the opened ACS Stream on which routing dialog is ending.

#### *eventClass*

This is a tag with the value **CSTAEVENTREPORT**, which identifies this message as a CSTA unsolicited

event.

***eventType***

This is a tag with the value **CSTA\_ROUTE\_END**, which identifies this message as a **CSTARouteEndEvent**.

***routeRegisterReqID***

This parameter 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 route register service request.

***routingCrossRefID***

This parameter contains the handle to the CSTA call routing dialog for this call. The application previously received this handle in the **CSTARouteRequestEvent** for the call.

***errorValue***

This parameter contains a cause code which giving the reason why the routing dialog ended.

***privateData***

If private data accompanies **CSTARouteEndEvent**, 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 **CSTARouteEndEvent** does not deliver private data to the application.

**Comments**

The switch sends **CSTARouteEndEvent** when a call has been successfully routed, cleared, or when the routing

server has failed to provide a route select within the switch's time limit. This event is unsolicited and can occur at any time.

## **cstaRouteEnd( )XE "cstaRouteEnd( )"**

The routing server (application) uses **cstaRouteEnd( )** to cancel an active routing dialog for a call. The service request includes a cause value giving the reason for the routing dialog termination.

### **Syntax**

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

RetCode_t cstaRouteEnd (
    ACSHandle_t      acsHandle,
    RouteRegisterReqID_t routeRegisterReqID,
    RoutingCrossRefID_t routingCrossRefID,
    CSTAUniversalFailure_t errorValue;
    PrivateData_t    *privateData);
```

### **Parameters**

#### ***acsHandle***

This is the handle for the opened ACS Stream on which the application is terminating a routing dialog for a call.

#### ***routeRegisterReqID***

This parameter 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 route register service request.

#### ***routingCrossRefID***

This parameter contains the handle to the CSTA call routing dialog for a call. The application previously received this handle in the **CSTARouteRequestEvent** for the call. This is the routing dialog that the application is ending.

#### ***errorValue***

The application supplies this cause code giving the

reason why it is ending the routing dialog.

***privateData***

This is an optional pointer to CSTA private data.

**Return Values**

**cstaRouteEnd( )** returns a non-negative value when successful.

The following are possible negative error conditions for this function:

***ACSERR\_BADHDL***

The application provided a bad or unknown ***acsHandle***.

***ACSERR\_STREAM\_FAILED***

A previously active ACS Stream has been abnormally aborted.

**Comments**

A routing server application can use **cstaRouteEnd( )** when it cannot route a call. This can occur if:

- ◆ the application receives a routing request for a call without sufficient call information and it cannot determine a routing destination.
- ◆ the application has already routed calls to all available destinations and those calls remain active at those destinations.
- ◆ the application does not have access to a database necessary to route the call

In these cases, the application uses **cstaRouteEnd( )** to inform the switch that it will not route the call in

question. **cstaRouteEnd()** will terminate the CSTA routing dialog (*routingCrossRefID*) for the call. **cstaRouteEnd()** does not clear the call. The switch will continue to process the call using whatever default routing algorithm is available (implementation specific).