

6 *Status Reporting Services* XE "Status Reporting:Services" §

TSAPI Status Services are those function calls and events that are related to unsolicited TSAPI event messages XE "Events:Unsolicited" §. External telephony activity on the switch, External telephony activity at a device, or human user activity can all cause unsolicited TSAPI events. Such events messages are asynchronous XE "Events:Asynchronous" § in nature. An application typically cannot anticipate their arrival. For example, an event informing the application of an incoming call to a device (e.g. a telephone station) is an unsolicited, asynchronous TSAPI event (since the application did not initiate the call and such a call can arrive at any time).

Applications use the status reporting request function to turn-on or turn-off status event reporting for a CSTA device (e.g. a desktop telephone). An application can use this function to turn-on/turn-off status reporting for any stations on the switch where monitoring is required (assuming proper access permissions are administered at the Telephony Server).

Status Reporting Functions XE "Status Reporting:Functions "§and Confirmation EventsXE "Status Reporting:Confirmation Events"§

Applications may use Event reporting to determine the changes in the state of a call or a connection associated with a device which is of interest to the application. This section describes the functions that an application uses to request unsolicited event reports for a telephony device or for calls. **In the case of calls, the application must first control the calls.**

Events provide the application with information about the state of calls or connection. The application may keep track of device or call states. If the application needs to maintain call state information for a specific device or call within the switch, it must establish a device or call "monitor" to keep track of the real-time state information for the call or device.

Applications should always be "event driven" and use events received from the Telephony Server react to changes in call or connection state rather using a specific switch implementation's call state mode. Following this guideline will simplify the support of applications across various switch implementations of TSAPI.

An application calls the **cstaMonitorDevice()**, **cstaMonitorCall()**, or **cstaMonitorCallsViaDevice()** function to initiate event reporting for a specific device or call. Event reporting can be provided for a device, a call, or for calls at a monitored device. An application can request two different types of event monitors using these functions. The monitor types are:

- ◆ **Call-type monitorXE "Call-type monitor"§** - call-type monitors provide monitoring (event reporting) for unsolicited events about a specific call from "cradle-to-grave". In other words, a call-type monitor provides events

for a specific call regardless of the devices at which the call may appear for the duration of the call.

Using call-type monitoring, an application can determine the current state of the call using the TSAPI events. For example, if a call monitor exists for a specific call and that call transfers or forwards to other devices, the sending device ceases to participate in the call, but event reporting continues (telling the application about the new devices participating in the call). Thus, a call-type monitor will provide call state information as the switch, other applications, and human users interact to route a call..

A switch may assign a new call identifier to a call as it is transferred or conferenced. The new call identifier will be provided in the event report associated with the conference or transfer function being requested by the controller of the call.

- ◆ **Device-type monitor**^{XE} "**Device-type monitor**"§ - device-type monitors provide the application with call or connection state information about calls at a specific device (the monitored device). TSAPI reports any events about the calls at the monitored device on a device-type monitor. If a call is transferred, drops, or forwards from the monitored device, TSAPI stops reporting events for that call.

If an application begins monitoring a device when call(s) are already in progress at the monitored device, TSAPI will not provide events for those calls. TSAPI provides events for calls that arrive at the device after it sends the confirmation event for the device monitor request.

An application may request a monitor on a call or a device but not both. Each monitor must be either a call or device monitor. An application may request multiple monitors, with various monitors being of various types. An application must setup

multiple monitors if it wants to monitor multiple devices or calls at the same time. The switch may impose limitations on the maximum number of simultaneous monitors which can exist for any given switch, call, or device. TSAPI does not place any such restrictions on the application.

When an application requests a device or call monitor, it can also specify an event filter. An event filter causes TSAPI to discard those events which the application is not interested in. An application may specify a filter when it establishes the monitor for a device or call. An application may also use the **cstaChangeMonitorFilter()** to change the filter after the monitor is active.

To receive events, an application must have an active ACS Stream and implement an event handling mechanism. Further, the reception of unsolicited events *XE "Events:Unsolicited" §* requires an active monitor. See the **Control Services** and **Status Reporting Services** sections for more information on events.

cstaMonitorDevice(XE "cstaMonitorDevice("§)

The Monitor Start service is used to initiate unsolicited event reporting for a device type monitoring on a device object. The unsolicited event reports will be provided for all endpoints within a CSTA switching sub-domain and optionally for endpoints outside of the CSTA switching sub-domain (implementation specific) which are involved with a monitored device.

Syntax

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

RetCode_t cstaMonitorDevice (
    ACSHandle_t      acsHandle,
    InvokeID_t       invokeID,
    DeviceID_t       *deviceID;CSTAMonitorFilter_t    *monitorFilter,
    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.

deviceID

Device ID of the device to be monitored.

monitorFilter

This parameter is used to specify a filter type to be used with the

object being monitored. Setting a bit to **true** in the *monitorFilter* structure causes the specific event to be **filtered out**, so the application will never see this event. Initialize the structure to all 0's to receive all types of monitor events. See `cstaMonitorDeviceConfEvent` for a definition of a `monitorFilter` structure.

privateData

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 **CSTAMonitorStartConfEvent** message to insure 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.

Comments

This function is used to start a device monitor on a CSTA device . The confirmation event for this function, i.e.

CSTAMonitorConfEvent will provide the application with the CSTA association handle to the monitored device or call, i.e. the Monitor Cross Reference Identifier (***monitorCrossRefID***) which defines the CSTA association on which the monitor will exist.

cstaMonitorCall()XE "cstaMonitorCall()"§

The Monitor Start service is used to initiate unsolicited event reporting for a call type monitoring on a call object. The unsolicited event reports will be provided for all endpoints within a CSTA switching sub-domain and optionally for endpoints outside of the CSTA switching sub-domain (implementation specific) which are involved with a monitored device.

Syntax

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

RetCode_t cstaMonitorCall (
    ACSHandle_t      acsHandle,
    InvokeID_t       invokeID,
    ConnectionID_t   *call,
    CSTAMonitorFilter_t *monitorFilter,
    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

Connection ID of the call to be monitored.

monitorFilter

This parameter is used to specify a filter type to be used with the object being monitored. Setting a bit to **true** in the ***monitorFilter*** structure causes the specific event to be **filtered out**, so the application will never see this event. Initialize the structure to all 0's to receive all types of monitor events. See `cstaMonitorDeviceConfEvent` for a definition of a `monitorFilter` structure.

privateData

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 **CSTAMonitorStartConfEvent** message to insure 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.

Comments

This function is used to start a call monitor on a CSTA device . The confirmation event for this function, i.e.

CSTAMonitorConfEvent will provide the application with the CSTA association handle to the monitored device or call, i.e. the Monitor Cross Reference Identifier (***monitorCrossRefID***) which defines the CSTA association on which the monitor will exist.

cstaMonitorCallsViaDevice()XE "cstaMonitorCallsViaDevice()"§

The Monitor Start service is used to initiate unsolicited event reporting for a call type monitoring on a device object. The unsolicited event reports will be provided for all endpoints within a CSTA switching sub-domain and optionally for endpoints outside of the CSTA switching sub-domain (implementation specific) which are involved with a monitored device.

Syntax

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

RetCode_t cstaMonitorCallsViaDevice (
    ACSHandle_t          acsHandle,
    InvokeID_t           invokeID,
    DeviceID_t           *deviceID,
    CSTAMonitorFilter_t  *monitorFilter,
    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.

device

The deviceID of the device for which call monitoring should be started.

monitorFilter

This parameter is used to specify a filter type to be used with the object being monitored. Setting a bit to **true** in the ***monitorFilter*** structure causes the specific event to be **filtered out**, so the application will never see this event. Initialize the structure to all 0's to receive all types of monitor events. See `cstaMonitorDeviceConfEvent` for a definition of a `monitorFilter` structure.

privateData

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 **CSTAMonitorStartConfEvent** message to insure 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.

Comments

This function is used to start a monitor on a CSTA object (a device or a call). The confirmation event for this function, i.e. **CSTACallMonitorStartConfEvent** will provide the application with the CSTA association handle to the monitored device or call, i.e. the Monitor Cross Reference Identifier (***monitorCrossRefID***) which defines the CSTA association on which the monitor will exist. There are two-types of Monitor Service: call-type and device-type.

CSTAMonitorConfEventXE "CSTAMonitorConfEvent"§

This event is in response to the **cstaMonitorDevice()**, **cstaMonitorCall** or **cstaMonitorCallsViaDevice** function and contains the association handle being assigned to the CSTA association being used for status reporting.

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
            {
                CSTAMonitorConfEvent_t   monitorStart;
            } u;
        } cstaConfirmation;
    } event;} CSTAEvent_t;

typedef struct CSTAMonitorConfEvent_t {
    CSTAMonitorCrossRefID_t   monitorCrossRefID;
    CSTAMonitorFilter_t   monitorFilter;
} CSTAMonitorConfEvent_t;

typedef long           CSTAMonitorCrossRefID_t;

typedef unsigned short  CSTACallFilter_t;#define           CF_CALL_CLEARED
0x8000#define           CF_CONFERENCED 0x4000#define
CF_CONNECTION_CLEARED 0x2000#define           CF_DELIVERED
0x1000#define           CF_DIVERTED 0x0800#define
CF_ESTABLISHED 0x0400#define           CF_FAILED 0x0200#define
CF_HELD 0x0100#define           CF_NETWORK_REACHED 0x0080#define
CF_ORIGINATED 0x0040#define           CF_QUEUED 0x0020#define
CF_RETRIEVED 0x0010#define           CF_SERVICE_INITIATED 0x0008#define
CF_TRANSFERRED 0x0004typedef unsigned char  CSTAFeatureFilter_t;#define
FF_CALL_INFORMATION 0x80#define           FF_DO_NOT_DISTURB
0x40#define           FF_FORWARDING 0x20#define
FF_MESSAGE_WAITING 0x10typedef unsigned char  CSTAAgentFilter_t;#define
```

```

AF_LOGGED_ON 0x80#define          AF_LOGGED_OFF 0x40#define
AF_NOT_READY 0x20#define          AF_READY 0x10#define
AF_WORK_NOT_READY 0x08#define     AF_WORK_READY 0x04typedef
unsigned char CSTAMaintenanceFilter_t;#define      MF_BACK_IN_SERVICE
0x80#define      MF_OUT_OF_SERVICE 0x40typedef struct
CSTAMonitorFilter_t { CSTACallFilter_t      call; CSTAFeatureFilter_t feature;
CSTAAGENTFilter_t      agent; CSTAMaintenanceFilter_t      maintenance; Boolean
private;} CSTAMonitorFilter_t;

```

Parameters

acsHandle

This is the handle for the 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_MONITOR_CONF**, which identifies this message as an **CSTAMonitorDeviceConfEvent**.

invokeID

This parameter specifies the requested instance of the function or event. It is used to match a specific functions call request with its confirmation events. Unsolicited events will have this parameter set to zero.

monitorCrossRefID

This parameter contains the handle to the CSTA association for which the requested monitor has been established. This handle is typically chosen by the switch and should be used by the application as a reference to a specific established association.

monitorFilter

This parameter is used to specify the filter type which is active on the object being monitored by the application. Possible classes of values are: **CALL_FILTER**, **FEATURE_FILTER**, **AGENT_FILTER**, **MAINTENANCE_FILTER**, and **PRIVATE_FILTER**.

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 should be check by the application to obtain the monitorCrossRefID being assigned by the switch and to insure that the event filter requested has been activated. The events informs the application which filters is active on the given CSTA association.

cstaMonitorStop()XE "cstaMonitorStop()"§

The Monitor Stop Service is used to cancel a previously registered Monitor Start Service on an existing CSTA monitor association, i.e. an active **monitorCrossRefID**.

Syntax

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

RetCode_t cstaMonitorStop (
    ACSHandle_t          acsHandle,
    InvokeID_t          invokeID,
    CSTAMonitorCrossRefID_t  monitorCrossRefID,
    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.

monitorCrossRefID

This parameter identifies the original CSTA monitor association for which unsolicited event monitoring is to be canceled. This identifier is provided as a result of a monitor start service request (**cstaMonitorStart()**) in a **CSTAMonitorStartConfEvent** for a call or device monitor within the switching domain.

privateData

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 **CSTAMonitorStopConfEvent** message to insure 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.

Comments

This function is used to cancel a previously registered monitor association on a CSTA object (a device or a call object). Once a confirmation event is issued for this function, i.e. a **CSTAMonitorStopConfEvent**; it will terminate the previously active monitoring association and thus end event reporting for the monitored call or device.

CSTAMonitorStopConfEventXE "CSTAMonitorStopConfEvent"

This event is in response to the **cstaMonitorStop()** function and provides the application with a confirmation that the monitor association has been canceled. Once this confirmation event is issued all event reporting for the specific monitoring association will be discontinued.

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;
        } cstaConfirmation;
    } event;} CSTAEvent_t;
```

Parameters

acsHandle

This is the handle for the 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_MONITOR_STOP_CONF**, which identifies this message as an **CSTAMonitorStopConfEvent**.

invokeID

This parameter specifies the requested instance of the function or event. It is used to match a specific functions call request with its confirmation events. Unsolicited events will have this parameter set to zero.

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 a cancellation of a CSTA monitoring association. After this event is issued by the Telephony Server, no further events will be sent to the application on the monitoring association (***monitorCrossRefID***) which was canceled.

cstaChangeMonitorFilter()XE "cstaChangeMonitorFilter()"§

This function is used to request a change in the filter options for CSTA event reporting for a specific CSTA association. It allows the application to specify for which event category the application wishes to receive events.

Syntax

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

RetCode_t CSTAChangeMonitorFilter (
    ACSHandle_t          acsHandle,
    InvokeID_t           invokeID,
    CSTAMonitorCrossRefID_t  monitorCrossRefID,
    CSTAMonitorFilter_t  *filterlist,
    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.

monitorCrossRefID

This parameter identifies the CSTA association (association handle) for which a change in event filtering is required. The association identifier is provided by the server/switch when the association is established.

filterlist

This parameter identifies the filter type being requested. Possible classes of values are CALL_FILTER, FEATURE_FILTER, AGENT_FILTER, MAINTENANCE_FILTER, and PRIVATE_FILTER. This parameter also identifies the events to be filtered.

privateData

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 **CSTChangeMonitorFilterConfEvent** message to insure 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.

Comments

The **cstaEventFilter()** function is used to inform the API Client Library and the server that only certain types of events are required. All events not requested will be filtered by the server and not provided to the application

**CSTACHangeMonitorFilterConfEventXE
"CSTACHangeMonitorFilterConfEvent"§**

This event occurs as a result of the **cstaEventFilter()** function and informs the application which event filter was set by the server.

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_tacsHandle;EventClass_t      eventClass;  EventType_t  eventType;
} ACSEventHeader_t;

typedef struct
{
    ACSEventHeader_t  eventHeader;
    union
    {
        struct
        {
            InvokeID_t  invokeID_t;
            union
            {
                CSTACHangeMonitorFilterConfEvent changeMonitorFilter;
            } u;
        }
    }
}
```

```

        } cstaConfirmation;
    } event;} CSTAEvent_t;
typedef struct CSTAChangeMonitorFilterConfEvent_t
{
    CSTAMonitorFiltler_t      filterlist;
}

```

Parameters

acsHandle

This is the handle for the ACS Stream.

eventClass

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

eventType

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

invokeID

This parameter specifies the requested instance of the function or event. It is used to match a specific functions call request with its confirmation events. Unsolicited events will have this parameter set to zero.

filterlist

This parameter identifies the filter type being requested. Possible classes of values are **CALL_FILTER**, **FEATURE_FILTER**, **AGENT_FILTER**, **MAINTENANCE_FILTER**, and **PRIVATE_FILTER**.

This parameter also identifies the events to be filtered.

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 should be checked by the application to insure that the event filter requested has been activated and which filters are already active on the given CSTA association.

CSTAMonitorEndedXE "CSTAMonitorEnded"§

This unsolicited indication is sent by the driver/switch to indicate to the application that the monitor associated with the *monitorCrossRefID* has been stopped.

Syntax

The following structure shows only the relevant portions of the unions for this message. See Sections *ACS Data Types* and *CSTA Data Types* 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
        {
            CSTAMonitorCrossRefID_t   monitorCrossRefID;
            union
            {
                CSTAMonitorEnded_t   monitorEnded;
            } u;
        } cstaUnsolicited;
    } event;} CSTAEvent_t;
typedef struct CSTAMonitorEndedEvent_t {
    CSTAEventCause_t cause;
} CSTAMonitorEndedEvent_t;
```

Parameters

acsHandle

This is the handle for the ACS Stream.

eventClass

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

eventType

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

which identifies this message as an **CSTAMonitorStopEvent**.

monitorCrossRefID,

This parameter contains the handle to the CSTA association for which this event is associated. This handle is typically chosen by the switch and should be used by the application as a reference to a specific established association.

cause

The cause code indicating the reason the monitor was stopped.

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 event is provided by the driver/switch when it can no longer provide the requested events associated with the *monitorCrossRefId*.

Call Event Reports (Unsolicited)XE "Events"§

This section defines the unsolicited TSAPI Event Reports that result from call activity at the Device or the switch. These events provide an application with call status information. Applications, users, and switch administrators may also use switch features that interact with monitored devices and calls, resulting in additional call events. One example of such a feature is call coverage paths.

CSTACallClearedEventXE "cstaCallClearedEvent"§

This event report indicates when a call is torn down. This can occur when the last device has disconnected from the call or when a call is dissolved by another party to the call - like a conference call being dissolved by the conference controller.

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
            {
                CSTAMonitorCrossRefID_t   monitorCrossRefID;
                CSTACallClearedEvent   callClear;
            } u;
        } cstaUnsolicited;
    } event;} CSTAEvent_t;

typedef enum LocalConnectionState_t {   CS_NULL = 0,
    CS_INITIATE = 1, CS_ALERTING = 2, CS_CONNECT = 3, CS_HOLD = 4,
    CS_QUEUED = 5, CS_FAIL = 6} LocalConnectionState_t;typedef enum
CSTAEventCause_t {   ACTIVE_MONITOR = 1, ALTERNATE = 2, BUSY = 3,
    CALL_BACK = 4, CALL_CANCELLED = 5, CALL_FORWARD_ALWAYS = 6,
    CALL_FORWARD_BUSY = 7, CALL_FORWARD_NO_ANSWER = 8,
    CALL_FORWARD = 9, CALL_NOT_ANSWERED = 10, CALL_PICKUP = 11,
    CAMP_ON = 12, DEST_NOT_OBTAINABLE = 13, DO_NOT_DISTURB = 14,
    INCOMPATIBLE_DESTINATION = 15, INVALID_ACCOUNT_CODE = 16,
    KEY_CONFERENCE = 17, LOCKOUT = 18, MAINTENANCE = 19,
    NETWORK_CONGESTION = 20, NETWORK_NOT_OBTAINABLE = 21,
    NEW_CALL = 22, NO_AVAILABLE_AGENTS = 23, OVERRIDE = 24, PARK =
    25, OVERFLOW = 26, RECALL = 27, REDIRECTED = 28, REORDER_TONE =
    29, RESOURCES_NOT_AVAILABLE = 30, SILENT_MONITOR = 31, TRANSFER
    = 32, TRUNKS_BUSY = 33, VOICE_UNIT_INITIATOR = 34} CSTAEventCause_t;

typedef struct
{
    ConnectionID_t   clearedCall;
    LocalConnectionState_t   localConnectionInfo;
```

```
    CSTAEventCause_t    cause;  
} CSTACallClearedEvent_t;
```

Parameters

acsHandle

This is the handle for the ACS Stream.

eventClass

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

eventType

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

monitorCrossRefID,

This parameter contains the handle to the CSTA association for which this event is associated. This handle is typically chosen by the switch and should be used by the application as a reference to a specific established association.

clearedCall

This parameter identifies the call which has been cleared.

localConnectionInfo

This parameter defines the local connection state of the call after it has been cleared. This could be null, initiated, alerting, connected, held, queued, or failed.

cause

This parameter contains the cause value which indicates the reason or explanation for the occurrence of this event. The possible events are defined by **CSTAEventCause_t**.

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 event is usually provided after the **cstaClearCall()** function has been called by the application. It can also occur, unsolicited, when another endpoint (device) clears a call and the device being monitored by the API is part of the call cleared by the another endpoint. The event is also generated when the last remaining device has disconnected from the call.

Figure 6-1

Call Cleared Event Report
"Call Cleared Event Report" f f \13§
µ §

CSTAConferencedEventXE "cstaConferencedEvent"§

This event report provides indication that two separate calls have been conferenced (merged) into a single. This occurs without either party being removed from the resulting call.

Syntax

The following structure shows only the relevant portions of the unions for this message. See *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
            {
                CSTAMonitorCrossRefID_t   monitorCrossRefID;
                CSTAConferencedEvent_t   conferenced;
            } u;
        } cstaUnsolicited;
    } event;} CSTAEvent_t;

typedef struct
{
    ConnectionID_t   primaryOldCall;
    ConnectionID_t   secondaryOldCall;
    SubjectDeviceID_t   confController;
    SubjectDeviceID_t   addedParty;
    ConnectionList_t   conferenceConnections;
    LocalConnectionState_t   localConnectionInfo;
    CSTAEventCause_t   cause;
} CSTAConferencedEvent_t;
```

Parameters

acsHandle

This is the handle for the ACS Stream.

eventClass

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

eventType

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

monitorCrossRefID

This parameter contains the handle to the CSTA association for which this event is associated. This handle is typically chosen by the switch and should be used by the application as a reference to a specific established association.

primaryOldCall

This parameter identifies the primary known call to be conferenced. This is usually the held call pending the conference.

secondaryOldCall

This parameter identifies the secondary call (e.g. the consultative call) which is to be conferenced. This is usually the active call which is to be conferenced to the held call pending the conference.

confController

This structure identifies the device which is controlling the conference. This is the device which setup the conference. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required.

addedParty

This parameter identifies the device which is being added to the conference. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required.

conferenceConnections

This is a list of connections (parties) on the call which resulted from the conference. The call ID may be different from either the primary or secondary old call (or both).

localConnectionInfo

This parameter defines the local connection state of the call after it has been conferenced. This could be null, initiated, alerting, connected, held, queued, or failed.

cause

This parameter contains the cause value which indicates the reason or explanation for the occurrence of this event. The possible events are defined by **CSTAEventCause_t**.

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 event provides information regarding a conference after it has been requested by the application using the **CSTAConferenceCall()** function or other endpoints on the switch. The changes in the call states are as follows:

Figure 6-2

Conferenced Event Report
"Conferenced Event Report" \f f \13§
μ §

CSTAConnectionClearedEventXE "CSTAConnectionClearedEvent"

This event report indicates that a device associated with a call disconnects from the call or is dropped from the call. The event does not indicate that a transferring device has left a call through the act of transferring that call.

Syntax

The following structure shows only the relevant portions of the unions for this message. See *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
            {
                CSTAMonitorCrossRefID_t    monitorCrossRefID;
                CSTAConnectionClearedEvent_t    connectionCleared;
            } u;
        } cstaUnsolicited;
    } event;
} CSTAEvent_t;

typedef struct
{
    ConnectionID_t    droppedConnection;
    SubjectDeviceID_t    releasingDevice;
    SubjectDeviceID_t    localConnectionInfo;
    CSTAEventCause_t    cause;
} CSTAConnectionClearedEvent_t;
```

Parameters

acsHandle

This is the handle for the ACS Stream.

eventClass

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

identifies this message as an CSTA unsolicited event.

eventType

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

monitorCrossRefID

This parameter contains the handle to the CSTA association for which this event is associated. This handle is typically chosen by the switch and should be used by the application as a reference to a specific established association.

droppedConnection

This parameter identifies the Connection which was dropped from the call as a result of a device dropping from the call.

releasingDevice

This parameter identifies the device which dropped the call.

localConnectionInfo

This parameter defines the local connection state of the call after the connection has been cleared. This could be null, initiated, alerting, connected, held, queued, or failed.

cause

This parameter contains the cause value which indicates the reason or explanation for the occurrence of this event. The possible events are defined by **CSTAEventCause_t**.

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 event is used to determine which device disconnects from a multi-party call. The deviceID identifies the devices which disconnected or was disconnected from the call. The LocalConnectionInfo defines the state of the call at the monitored device after the device has been dropped from the call.

Figure 6-3

Connection Cleared Event Report
"Connection Cleared Event Report" \f f \l3§
μ §

CSTADeliveredEventXE "CSTADeliveredEvent"§

This event report indicates that a call is alerting (e.g. ringing) at a specific device or that the server has detected that a call is alerting at a specific device.

Syntax

The following structure shows only the relevant portions of the unions for this message. See Chapter 4 *Data Types* and *CSTA Data Types* 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
        {
            CSTAMonitorCrossRefID_t    monitorCrossRefID;
            union
            {
                CSTADeliveredEvent_t    delivered;
            } u;
        } cstaUnsolicited;
    } event;
} CSTAEvent_t;

typedef struct
{
    ConnectionID_t        connection;
    SubjectDeviceID_t     alertingDevice;
    CallingDeviceID_t     callingDevice;
    CalledDeviceID_t      calledDevice;
    RedirectionDevice_t   lastRedirectionDevice;
    LocalConnectionState_t    localConnectionInfo;
    CSTAEventCause_t       cause;
} CSTADeliveredEvent_t;
```

Parameters

acsHandle

This is the handle for the ACS Stream.

eventClass

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

identifies this message as an CSTA unsolicited event.

eventType

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

monitorCrossRefID

This parameter contains the handle to the CSTA association for which this event is associated. This handle is typically chosen by the switch and should be used by the application as a reference to a specific established association.

connection

This parameter identifies the Connection which is alerting

alertingDevice

This parameter indicates which device is alerting. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required.

callingDevice

This parameter identifies the calling device. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required

calledDevice

This parameter identifies the originally called device. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required

lastRedirectionDevice

This parameter will identify the previously alerted device in cases where the call was redirected or diverted to the alerting device. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required.

localConnectionInfo

This parameter defines the local connection state of the call after the Connection has alerted. This could be null, initiated, alerting, connected, held, queued, or failed.

cause

This parameter contains the cause value which indicates the reason or explanation for the occurrence of this event. The possible events are defined by **CSTAEventCause_t**.

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 event provides all the necessary information required when a new call arrives at a device. This will include the calling and called numbers.

Figure 6-4

Delivered Event Report
"Delivered Event Report" \f f \13§
μ §

CSTADivertedEventXE "CSTADivertedEvent"§

This event report identifies a call which has been deflected or diverted from a monitored device. The call is no longer present or associated with the device.

Syntax

The following structure shows only the relevant portions of the unions for this message. See *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
        {
            CSTAMonitorCrossRefID_t  monitorCrossRefID;
            union
            {
                CSTADivertedEvent_t  diverted;
            } u;
        } cstaUnsolicited;
    } event;
} CSTAEvent_t;

typedef struct
{
    ConnectionID_t      connection;
    SubjectDeviceID_t   divertingDevice;
    CalledDeviceID_t    newDestination;
    LocalConnectionState_t  localConnectionInfo;
    CSTAEventCause_t    cause;
} CSTADivertedEvent_t;
```

Parameters

acsHandle

This is the handle for the ACS Stream.

eventClass

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

eventType

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

monitorCrossRefID

This parameter contains the handle to the CSTA association for which this event is associated. This handle is typically chosen by the switch and should be used by the application as a reference to a specific established association.

connection

This parameter indicates the Connection which was previously alerting. This can be the intended Connection for the call before it was diverted.

divertingDevice

This parameter indicates the device from which the call was diverted. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required.

newDestination

This parameter indicates the device to which the call was diverted. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required.

localConnectionInfo

This parameter defines the local connection state of the device being monitored. This could be null, initiated, alerting, connected, held, queued, or failed.

cause

This parameter contains the cause value which indicates the reason or explanation for the occurrence of this event. The possible events are defined by **CSTAEventCause_t**.

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 event is used to determine information about a call which has been diverted from a monitored device. This includes information on which device the call is being diverted.

Figure 6-5

Diverted Event Report: "Diverted Event Report" \f f \l3§
µ §

CSTAEstablishedEventXE "CSTAEstablishedEvent"§

This event report identifies a call which has been deflected or diverted from a monitored device. The call is no longer present or associated with the device.

Syntax

The following structure shows only the relevant portions of the unions for this message. See *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
        {
            CSTAMonitorCrossRefID_t    monitorCrossRefID;
            union
            {
                CSTAEstablishedEvent_t    established;
            } u;
        } cstaUnsolicited;
    } event;
} CSTAEvent_t;

typedef struct
{
    ConnectionID_t        establishedConnection;
    SubjectDeviceID_t     answeringDevice;
    CallingDeviceID_t     callingDevice;
    CalledDeviceID_t      calledDevice;
    REDirectionDeviceID_t lastRedirectionDevice;
    LocalConnectionState_t localConnectionInfo;
    CSTAEventCause_t      cause;
} CSTAEstablishedEvent_t;
```

Parameters

acsHandle

This is the handle for the ACS Stream.

eventClass

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

eventType

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

monitorCrossRefID

This parameter contains the handle to the CSTA association for which this event is associated. This handle is typically chosen by the switch and should be used by the application as a reference to a specific established association.

establishedConnection

This parameter identifies the Connection which joined the call as a result of answering the call.

answeringDevice

This parameter indicates the device which has joined the call, i.e. the answering device. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required.

callingDevice

This indicates which device made the call, i.e. the calling device. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required.

calledDevice

This parameter indicates the originally called device. This may not always be the device answering a call as is the case with call forwarding or coverage, i.e. call redirection. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required.

lastRedirectionDevice

This parameter indicates the previously alerted device in cases where a call is redirected. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required.

localConnectionInfo

This parameter defines the local connection state of the device for the call which has been established. This could be null, initiated, alerting, connected, held, queued, or failed.

cause

This parameter contains the cause value which indicates the reason or explanation for the occurrence of this event. The possible events are defined by **CSTAEventCause_t**.

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 event is typically used to determine when a call is answered by an endpoint being called by the application. This includes the calling and called number identification.

Figure 6-6

Established Event Report: "Established Event Report" \f \l3§
μ §

CSTAFailedEventXE "CSTAFailedEvent"§

This event report indicates that a call cannot be completed. The event applies only to a single Connection.

Syntax

The following structure shows only the relevant portions of the unions for this message. See *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
        {
            CSTAMonitorCrossRefID_t    monitorCrossRefID;
            union
            {
                CSTAFailedEvent_t    failed;
            } u;
        } cstaUnsolicited;
    } event;
} CSTAEvent_t;

typedef struct
{
    ConnectionID_t    failedConnection;
    SubjectDeviceID_t    failingDevice;
    CalledDeviceID_t    calledDevice;
    LocalConnectionState_t    localConnectionInfo;
    CSTAEventCause_t    cause;
} CSTAFailedEvent_t;
```

Parameters

acsHandle

This is the handle for the ACS Stream.

eventClass

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

eventType

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

monitorCrossRefID

This parameter contains the handle to the CSTA association for which this event is associated. This handle is typically chosen by the switch and should be used by the application as a reference to a specific established association.

failedConnection

This parameter indicates which Connection has failed.

failingDevice

This parameter indicates which device has failed. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required.

calledDevice

This parameter indicates which device was called when the call failed. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required.

localConnectionInfo

This parameter defines the local connection state of the call after the Connection has failed. This could be null, initiated, alerting, connected, held, queued, or failed.

cause

This parameter contains the cause value which indicates the reason or explanation for the occurrence of this event. The possible events are defined by **CSTAEventCause_t**.

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 event occurs anytime a call cannot be completed for any reason (e.g. Stations Busy, Reorder Tone, Trunks Busy, etc.). The **cause** parameter contains the reason why the call failed.

Figure 6-7

Failed Event Report
Failed Event Report
μ §

CSTAHeldEventXE "CSTAHeldEvent"§

This event report indicates that the server has detected that communications on a particular Connection has be interrupted (i.e. put on hold) by one of the devices on the call. This event is usually associated with a call being placed on hold at a device.

Syntax

The following structure shows only the relevant portions of the unions for this message. See **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
        {
            CSTAMonitorCrossRefID_t    monitorCrossRefID;
            union
            {
                CSTAHeldEvent_t    held;
            } u;
        } cstaUnsolicited;
    } event;
} CSTAEvent_t;

typedef struct
{
    ConnectionID_t        heldConnection;
    SubjectDeviceID_t     holdingDevice;
    LocalConnectionState_t    localConnectionInfo;
    CSTAEventCause_t      cause;
} CSTAHeldEvent_t;
```

Parameters

acsHandle

This is the handle for the ACS Stream.

eventClass

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

identifies this message as an CSTA unsolicited event.

eventType

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

monitorCrossRefID

This parameter contains the handle to the CSTA association for which this event is associated. This handle is typically chosen by the switch and should be used by the application as a reference to a specific established association.

heldConnection

This parameter identifies the Connection which was put on hold by the device.

holdingDevice

This parameter identifies the device which placed the connection on hold. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required.

localConnectionInfo

This parameter defines the local connection state of the call after the Connection has been put on hold. This could be null, initiated, alerting, connected, held, queued, or failed.

cause

This parameter contains the cause value which indicates the reason or explanation for the occurrence of this event. The possible events are defined by **CSTAEventCause_t**.

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 event occurs after a call has been placed on hold at a specific device. This informs the application what device placed the connection on hold.

Figure 6-8

Held Event Report
"Held Event Report" \f f \l3§
μ §

CSTANetworkReachedEventXE "CSTANetworkReachedEvent"§

This event report informs the application that a call has left the switch on an outbound trunk and is being routed through the telephone network.

Syntax

The following structure shows only the relevant portions of the unions for this message. See *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
        {
            CSTAMonitorCrossRefID_t    monitorCrossRefID;
            union
            {
                CSTANetworkReachedEvent_t    networkReached;
            } u;
        } cstaUnsolicited;
    } event;
} CSTAEvent_t;

typedef struct
{
    ConnectionID_t        connection;
    SubjectDeviceID_t     trunkUsed;
    CalledDeviceID_t      calledDevice;
    LocalConnectionState_t    localConnectionInfo;
    CSTAEventCause_t      cause;
} CSTAHeldEvent_t;
```

Parameters

acsHandle

This is the handle for the ACS Stream.

eventClass

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

eventType

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

monitorCrossRefID

This parameter contains the handle to the CSTA association for which this event is associated. This handle is typically chosen by the switch and should be used by the application as a reference to a specific established association.

connection

This parameter specifies the Connection ID for the outbound connection associated with the trunk and its connection to the network (see figure below).

trunkUsed

This parameter specifies the trunk that was used to establish the Connection with the telephone network. If the device (i.e. the trunk) is not specified, then the parameter will indicate that the device was not known or that it was not required.

calledDevice

This parameter indicates the destination device for the call. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required.

localConnectionInfo

This parameter defines the local connection state of the call after the Connection has cut-through into the telephone network. This could be null, initiated, alerting, connected, held, queued, or failed.

cause

This parameter contains the cause value which indicates the reason or explanation for the occurrence of this event. The possible events are defined by **CSTAEventCause_t**.

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

Once this event occurs the level of call related status information may decrease depending on the type of trunk being used to route the call to it's destination across the telephone network. The amount of call related status information provided by the network will depend on the type of trunk and telephone network being used to complete the call. Call status information may be limited to the disconnect or drop event. This only applies for calls to other network endpoints and not to calls within the switch being controlled by the server.

Figure 6-9

Network Reached Event Report
"Network Reached Event Report" \f f \13§
μ §

CSTAOriginatedEventXE "CSTAOriginatedEvent"\$

This event report informs the application that the switch is attempting to establish a call as a result of a completed request from the application.

Syntax

The following structure shows only the relevant portions of the unions for this message. See *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
        {
            CSTAMonitorCrossRefID_t    monitorCrossRefID;
            union
            {
                CSTAOriginatedEvent_t    originated;
            } u;
        } cstaUnsolicited;
    } event;
} CSTAEvent_t;

typedef struct
{
    ConnectionID_t        originatedConnection;
    SubjectDeviceID_t     callingDevice;
    CalledDeviceID_t      calledDevice;
    LocalConnectionState_t    localConnectionInfo;
    CSTAEventCause_t      cause;
} CSTAOriginatedEvent_t;
```

Parameters

acsHandle

This is the handle for the ACS Stream.

eventClass

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

eventType

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

monitorCrossRefID

This parameter contains the handle to the CSTA association for which this event is associated. This handle is typically chosen by the switch and should be used by the application as a reference to a specific established association.

originatedConnection

This parameter identifies the Connection where a call has been originated.

callingDevice

This parameter identifies the device from which the call has been originated. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required.

calledDevice

This parameter identifies the device for which the originated call is intended. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required.

localConnectionInfo

This parameter defines the local connection state of the call after the Connection has been originated. This could be null, initiated, alerting, connected, held, queued, or failed.

cause

This parameter contains the cause value which indicates the reason or explanation for the occurrence of this event. The possible events are defined by **CSTAEventCause_t**.

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 event indicates that a call is being launched by the switch on behalf of the request from the application. The event only indicates that the switch is attempting to make the call. The application should check for additional events to determine the status of the call as it proceeds either through the switch or out to the telephone network.

Figure 6-10

Originated Event Report
"Originated Event Report" \f \13§
µ §

CSTAQueuedEventXE "CSTAQueuedEvent"§

This event report indicates that a call has been queued to an ACD Split, a hunt group, or others devices which support call queues. Call can also be queued during network re-routing without specifying a device.

Syntax

The following structure shows only the relevant portions of the unions for this message. See **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
        {
            CSTAMonitorCrossRefID_t   monitorCrossRefID;
            union
            {
                CSTAQueuedEvent_t   queued;
            } u;
        } cstaUnsolicited;
    } event;} CSTAEvent_t;

typedef struct
{
    ConnectionID_t   queuedConnection;
    SubjectDeviceID_t   queue;
    SubjectDeviceID_t   callingDevice;
    CalledDeviceID_t   calledDevice;
    RedirectionDeviceID_t   lastRedirectionDevice;
    int   numberQueued;
    LocalConnectionState_t   localConnectionInfo;
    CSTAEventCause_t   cause;
} CSTAQueuedEvent_t;
```

Parameters

acsHandle

This is the handle for the ACS Stream.

eventClass

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

eventType

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

monitorCrossRefID

This parameter contains the handle to the CSTA association for which this event is associated. This handle is typically chosen by the switch and should be used by the application as a reference to a specific established association.

queuedConnection

This indicates the Connection was queued to the device.

queue

This parameter specifies the device to which the call has been queued. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required.

callingDevice

This parameter indicates the device who queued the call. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required.

calledDevice

This parameter indicates the device which was called (the intended recipient of the call). If the device is not specified, then the parameter will indicate that the device was not known or that it was not required.

lastRedirectionDevice

This parameter identifies the last device which redirected the call, if the call has been redirected. If the device is not specified,

then the parameter will indicate that the device was not known or that it was not required.

numberQueued

This parameter indicates how many calls are queued to the queuing device.

localConnectionInfo

This parameter defines the local connection state of the call after the call has been queued. This could be null, initiated, alerting, connected, held, queued, or failed.

cause

This parameter contains the cause value which indicates the reason or explanation for the occurrence of this event. The possible events are defined by **CSTAEventCause_t**.

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 event usually occurs when an application is monitoring a call, a Vector Directory Number (VDN), an ACD Split, or a hunt group. The event also provides information pertaining to the number of calls that have been queued to a device. This information can be useful to applications managing the queue at the device.

Figure 6-11

Queued Event Reporttc "Queued Event Report" \f f \13§

μ §