Chapter

9 *Escape and Maintenance Services*

This chapter describes the CSTA Escape and Maintenance Services.

Escape Services

XE "Escape Service"§Switching domains use Escape Services to enhance TSAPI functions with "private" services which are specific to the switch or PBX Driver implementation (see Section 2.3). Each switch vendor may define functions within the CSTA private services framework, even though CSTA does not incorporate these services. Although the functions defined within escape services can vary from one implementation to the next, the way the application accesses these functions is consistent. Escape Services use the same programming model as all other CSTA services. Figure 9-1 illustrates this model.

When an application requests an escape service from a server it receives a confirmation event or a Universal Failure in the same fashion as for other TSAPI services. The Escape Service request parameters are an ***acsHandle*** (to the open stream), an ***invokeID*** and a private data parameter. The confirmation event includes the ***acsHandle***, the ***invokeID***, and the private data response.

Escape Services also includes an unsolicited private event which

a server can send to an application at any time a CSTA monitor association exists on a CSTA call or device object (see Section 6 - **cstaMonitorStart**( )).

Figure 9-2

Escape Services Model XE "Escape Service:Model"§tc "Escape Services Model" \f f \l3§

Applications can also send Escape Services requests to a switch. For most CSTA services the application is always a client in the computing domain. However, an escape service could operate in the opposite direction (such as routing does). Although the client/server role may change, services are always uni-directional where either the switch or application is always the requester for a service.

TSAPI includes escape service definitions for both the "*Application as the Client*" and the "*Switch as the Client*".

See vendor specific documentation for more information on what, if any, Escape Services are supported by a specific vendor. Escape Service Functions are generally not portable across different vendor implementations. Some implementations may support Escape Services either bi-directionally or unidirectional (one-way only) depending on the needs and capabilities of the switch driver

Maintenance Services

XE "Maintenance Services"§There are two different types of CSTA maintenance services:

- device status maintenance events which provide status information for device objects, and
- bi-directional system status maintenance services which provide information on the overall status of the system.

The device status events inform the application when the switch places a monitored device in or out of service. When a device object is removed from service, the application may monitor the device (e.g. **cstaMonitorStart**( ) or **cstaDevSnapshotReq**( )) but may not request services for that device. For example, an application request for a **cstaMakeCall**( ) returns an error when the device is out of service.

System Status services inform applications or switches about the status of the switching or computing domains, respectively. Table 9-1 shows the System Status Services' system status information (cause codes).

Table 9-2

System Status Cause Codes XE "System status:Cause codes"§tc "System Status Cause Codes"\f t \l3§

System Status Cause Code	Cause Code Definitions
Initializing	the system is re-initializing or restarting. This status indicates that the system is temporarily unable to respond to any requests. If provided, this status message shall be followed by an Enable status message that indicates that the initialization process is completed.
Enabled	request and responses are enabled, usually after a disruption or restart. This status indication shall be sent after an initializing status indicator has been sent and my be sent under other conditions. This status indicates that there are no outstanding monitor requests.
Normal	a System Status Event with this cause value can be sent at any time to indicate that the status is normal. This status has no effect on other services.
Message Lost	this status indicates that a request, response, or event report may have been lost.
Disabled	this cause value indicates that active **cstaMonitorStart**( ) monitor requests via have been disabled. Other requests and responses may also be disabled, but, unlike monitors, reject responses are provided for those.
Overload Imminent	the system (driver, switch, or application) is about to reach an overload condition. The "client" should shed load to remedy the situation.

Overload Reached	the system (driver, switch, or application) has reached an overload condition and may take action to shed load. The server (the application, driver, or switch) may then take action to decrease message traffic. This may include stopping existing monitors or rejecting any new requests sent by the client.
Overload Relieved	the system (driver, switch, or application) has determined that the overload condition has passed and normal application operation may resume.

The System Status services are bi-directional and thus can originate at the application domain or at the driver/switch domain. Figure 9-3 shows System Status Maintenance Services.

Figure 9-4

System Status Maintenance ServicesXE "System status:Maintenance services"§tc "System Status Maintenance Services" \f f \l3§

An application can obtain System Status information in one of two different ways :

- the client can ask for the information using a request to the "server" and obtain the information in a confirmation event, or
- the client can register for System Status messages and receive unsolicited events containing system status changes.

A switch or application may issue the System Status request

(**cstaSysStatReq**( )) to obtain status information from the "server" (the application or switch, respectively, depending on the direction of the request). A System Status response (**CSTASysStatReqConfEvent**) provides the "client" with the current system status information for the "server". The "server" may send unsolicited events can be sent to the client if the client used the **cstaSysStatStart**( ) service to register for System Status events. The System Status unsolicited event (**CSTASysStatEvent**) is the same in structure as the confirmation event (**CSTASysStatReqConfEvent**) except that the "server" sends it to the "client" automatically.

Escape Services : Application as Client

XE "Escape Service:Application as Client"§This section defines escape services for situation where the application is the service requester in the client/server relationship (see Figure 9-5). The services include an escape service request, a confirmation event to the request, and an unsolicited escape service event that originates at the driver or switching domain.

cstaEscapeService( )XE "cstaEscapeService( )"§

This service allows the application to request a service which is not defined by the CSTA Standard but rather by a switch vendor. A service request made by this function will be specific to an implementation.

Syntax

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

RetCode_t   cstaEscapeService (
      ACSHandle_t          acsHandle,
      InvokeID_t           invokeID,
      PrivateData_t        *privateData);
```

Parameters

acsHandle
This is the handle to an active 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 valid 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.

privateData
This is a pointer to the CSTA private data extension mechanism. This parameter in NOT optional for this function and must be passed by the application. If the parameter is NULL an error will be returned to the application and the API Client Library Driver will reject the service request.

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 **CSTAEscapeServiceConfEvent** 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_NOCONN
This return value indicates that a previously active ACS Stream has been abnormally aborted.

ACSERR_REQDENIED
This return value indicates that a ACS Stream is established but a requested capability has been

denied by the Client Library Software Driver.

ACSERR_NULLPARAMETER
This error indicates that the pointer to the CSTA Private Data information is NULL and thus no private data is available to send to the driver/switch. No action is taken by the API Client Library Driver.

Comments

This function is used to send private data information to the driver/switch.

CSTAEscapeServiceConfEventXE "CSTAEscapeServiceConfEvent"§

This confirmation event is sent in response to the **cstaEscapeService**( ) service and provides the positive acknowledgment to the request. The event includes any private information that is to be provided as part of a confirmation event to the service request.

Syntax

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

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

typedef struct
{
                ACSEventHeader_t    eventHeader;
        union
        {               struct
                {                   InvokeID_t        invokeID;         } cstaConfirmation;
        } event;} CSTAEvent_t;
```

Parameters

acsHandle
This is the handle for the opened ACS Stream.

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

eventType
This is a tag with the value **CSTA_ESCAPE_SERVICE_CONF** which identifies

this message as an **CSTAEscapeServiceConfEvent.**

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

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

Comments

This event always occurs as a result of a normal (positive) service request made through the **cstaEscapeService**( ) service. The information contained in the ***privateData*** parameter is implementation specific.

CSTAPrivateEventXE "CSTAPrivateEvent"§

This event report allows for unsolicited, implementation specific event reporting. The informational contents of this event will be defined by a specific implementation.

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
                        {
                                CSTAPrivateEvent_t  privateData;
                        } u;
                } cstaEventReport;

        } event;} CSTAEvent_t;
```

Parameters

acsHandle
This is the handle for the opened ACS Stream.

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_PRIVATE**, which identifies this message as an **CSTAPrivateEvent.**

monitorCrossRefID
Does not apply to this event.

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 for providing unsolicited, implementation specific event information. This event can occur at any time and does not have be specific to a monitored object. The event can be sent by the driver/switch even though the application does not have a monitored object. When a monitor exists, the **PrivateStatusEvent** is used by the driver/switch to send private status information pertaining to a monitored object. The **PrivateEvent** is used for all other cases of unsolicited private events and is not associated with a monitoring association.

CSTAPrivateStatusEventXE "CSTAPrivateStatusEvent"§

This event report allows for unsolicited, implementation specific event reporting for a monitored object. The informational contents of this event will be defined by a specific implementation.

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
            {                 CSTAMonitorCrossRefID_t       monitorCrossRefID;
            } cstaUnsolicited;

      } event;} CSTAEvent_t;
```

Parameters

acsHandle
This is the handle for the opened 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_PRIVATE_STATUS**, which identifies this message as an **CSTAPrivateStatusEvent.**

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.

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 for providing implementation specific event information which is not defined in any other event in the API. The event is always used for private information on a monitoring association. A monitor must be active (**cstaMonitorStart**( )) before this event can be sent to the application by the driver/switch. This event is always sent from the driver/switch to the application and it is not bi-directional.

Escape Service : Driver/Switch as the ClientXE "Escape Service:Driver/Switch as the Client"§

This section defines escape services for cases where the Driver/Switch is the service requester in the client/server relationship (see Figure 9-1). The services include an escape service request event, a confirmation function for the request, and an unsolicited escape service event that originates at the application domain.

CSTAEscapeServiceReqXE "CSTAEscapeServiceReq"§

This unsolicited event is sent by the driver/switch to request a private service from the application. The event includes the service request as private information for which the application must provide a positive or negative acknowledgment.

Syntax

The following structure shows only the relevant portions of the unions for this message. See ***4.3 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
                        {
                              CSTAEscapeSvcReqEvent_t       escapeSvcReqeust;
                        } u;
            } cstaRequestEvent;

      } event;} CSTAEvent_t;
```

Parameters

acsHandle
This is the handle for the opened ACS Stream.

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

eventType
This is a tag with the value **CSTA_ESCAPE_SVC_REQ**, which identifies this

message as an **CSTAEscapeServiceReq**

invokeID
This parameter defines the invoke identifier selected by the driver/switch for the specific private request. This parameter *must* be returned, unchanged, in the response to this request in order for the driver/switch to match a service request with a service response.

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 sent by the driver/switch to request an escape or private service when the application is providing the "server" function in the client/server relationship. The response to this event will be accomplished via the **cstaEscapeServiceConf**( ) service.

cstaEscapeServiceConf( )XE "cstaEscapeServiceConf( )"§

This service allows the application to respond to a **CSTAEscapeServiceEvent** which originated at the driver/switch. A service response made by this function will be specific to an implementation.

Syntax

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

RetCode_t   cstaEscapeServiceConf (
      ACSHandle_t              acsHandle,
      InvokeID_t               invokeID,
      CSTAUniversalFailure_t         error,       /* negative ACK */
      PrivateData_t            *privateData),   /* positive ACK */
```

Parameters

acsHandle
This is the handle to an active ACS Stream.

invokeID
The invoke identifier used in this function *must* be the same value (unchanged) as that provided in the **CSTAEscapeServiceReq** for which this services request is being called. The same ***invokeID*** value must be used in order for the driver/switch to match the instances of a previous service event and the service confirmation to that event provided by this function call.

error
This parameter is used to provide a negative acknowledgment to the **CSTAEscapeServiceReq**. See **CSTAUniversalFailureConfEvent** for a definition of the possible error values for this parameter. If the ***error*** pointer is NULL this will indicates that the event contains a positive acknowledgment.

privateData
This is a pointer to the CSTA private data extension mechanism which contains the positive acknowledgment to the CSTAEscapeServiceEvent. If the private pointer is NULL this will indicate that the event contains a negative acknowledgment.

Return Values

This function never returns an invoke identifier since there is no confirmation event for this service. The function does return errors conditions during the processing of the request by the API Client Library. A return value of zero (0) indicates that the request has been accepted by the Library. This function never returns a positive value.

Possible local error returns are (negative returns):

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

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

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

ACSERR_NONULL
This error indicates that neither the ***error*** or ***privateData*** pointers are NULL. One of these pointers *must be* NULL to indicate either a positive or negative acknowledgment to the request. No action is taken by the API Client

Library.

ACSERR_NORESPONSE
This error indicates that both the ***error*** or the ***privateData*** pointer are NULL. In this case the API Client Library has nothing to send to the driver/switch and rejects the response. The request associated with the invoke identifier from the driver/server will still be outstanding and the application must respond by calling this function with acceptable parameters.

ACSERR_BADINVOKEID
This error indicates that the invoke identifier being returned by the application is not one that is outstanding from the driver/switch. The API Client Library will keep track of the driver/switch-based invoke id's until the application responds to the specific request from the driver/switch.

Comments

This function is used to send a response to a private request from the driver/switch. The event supports both a positive and negative acknowledgment to the request. One of the two pointers (***error*** or ***privateData***) must be NULL in order for the request to be successfully processed by the API Client Library. This would indicate a positive or negative acknowledgment to the request made by the driver/switch.

cstaSendPrivateEvent( )XE "cstaSendPrivateEvent( )"§

This service allows the application to send an unsolicited private event to the driver/switch. An event sent by this function will be specific to an implementation.

Syntax

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

RetCode_t   cstaSendPrivateEvent (
      ACSHandle_t                acsHandle,
      PrivateData_t          *privateData),
```

Parameters

acsHandle
This is the handle to an active ACS Stream.

privateData
This is a pointer to the CSTA private data extension mechanism. This parameter in NOT optional for this function and must be passed by the application. If the parameter is NULL an error will be returned to the application and the API Client Library Driver will reject the service request.

Return Values

This function never returns an invoke identifier since there is no confirmation event for this service. The function does return errors conditions during the processing of the request by the API Client Library. A return value of zero (0) indicates that the request has been accepted by the Library. This function never returns a positive value.

Possible local error returns are (negative returns):

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

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

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

ACSERR_NULLPARAMETER
This error indicates that the pointer to the CSTA Private Data information is NULL and thus no private data is available to send to the driver/switch. No action is taken by the API Client Library Driver.

Comments

This function is used to send unsolicited, private data information to the driver/switch when the application is supporting the "server" role in the client/server relationship.

Maintenance Services: Device StatusXE "Maintenance Services"§

This section describes the CSTA Maintenance Services which provide device status information. To receive device status information, an application must monitor the device(e.g. the application must have an active ***monitorCrossRefID*** for the device). These events are unidirectional and always originate in the switch domain.

CSTABackInServiceEventXE "CSTABackInServiceEvent"§

This event report indicates that a monitored device object has returned to services and operates normally.

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
                {                       CSTAMonitorCrossRefID_t       monitorCrossRefID;
                        union
                        {
                                CSTABackInServiceEvent_t  backInService;
                        } u;
                } cstaUnsolicited;
        } event;} CSTAEvent_t;
typedef struct
{
        DeviceID_t                      device;
        CSTAEventCause_t    case;
} CSTABackInServiceEvent_t;
```

Parameters

acsHandle
This is the handle for the opened 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_BACK_IN_SERVICE** , which identifies this message as an **CSTABackInServiceEvent.**

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.

device
Specifies the device which is back in service. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required.

cause
This parameter indicates the reason or explanation for the occurrence of this event. See Section 6 for more information.

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 previously inactive device (a device which is out service) has resumed normal operation. Once this event has occurred the application can then initiate an active service request (e.g. **cstaMakeCall**( )) for that specific device. A passive

service request can be done while a device is out of service, i.e. monitoring or Snapshot Services.

CSTAOutOfServiceEventXE "CSTAOutOfServiceEvent"§

This event report indicates that a monitored device object has entered a maintenance state and can no longer accept calls or be actively manipulated by the application.

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
            {                 CSTAMonitorCrossRefID_t       monitorCrossRefID;
                  union
                  {
                        CSTAOutOfServiceEvent_t  outOfService;
                  } u;
            } cstaUnsolicited;
      } event;} CSTAEvent_t;
typedef struct
{
      DeviceID_t                  device;
      CSTAEventCause_t    case;
} CSTAOutOfServiceEvent_t;
```

Parameters

acsHandle
This is the handle for the opened 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_OUT_OF_SERVICE** , which identifies this message as an **CSTAOutOfServiceEvent.**

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.

device
This parameter indicates the device which has been taken out of service. If the device is not specified, then the parameter will indicate that the device was not known or that it was not required.

cause
This parameter indicates the reason or explanation for the occurrence of this event. See Section 6 for more information.

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 previously active device (a device which is in service) has entered into a maintenance state, i.e. the device has been taken out of service. Once this event has occurred the application *can*

not initiate any new active service request (e.g. **cstaMakeCall**( )) for that specific device. A passive service request (e.g. monitoring or Snapshot Services) can be done while a device is out of service.

System Status - Application as the ClientXE "System status:Application as the client"§

This section defines the services which provide system level status information to the application or the driver/switch. The System Status service is bi-directional and thus the client/server relationship (see Figure 9-2) can be reversed.

cstaSysStatReq( )

This service allows the application to request system status information from the driver/switch domain.XE "cstaSysStatReq( )"§

Syntax

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

RetCode_t   cstaSysStatReq (
      ACSHandle_t              acsHandle,
      InvokeID_t               invokeID,
      PrivateData_t        *privateData);
```

Parameters

acsHandle
This is the handle to an active 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 valid 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.

privateData
This is a pointer to the CSTA private data extension mechanism. This is optional.

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 **CSTASysStatReqConfEvent** 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_NOCONN
This return value indicates that a previously active ACS Stream has been abnormally aborted.

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

Comments

This function is used to request the current overall system status for the driver/switch.

CSTASysStatReqConfEventXE "CSTASysStatReqConfEvent"§

This event is in response to the **cstaSysStatReq**( ) service and informs the application of the overall system status of the driver/switch.

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
                  {
                        CSTASysStatReqConfEvent  sysStatReq;
                  } u;
            } cstaConfirmation;
      } event;} CSTAEvent_t;
typedef struct CSTASysStatReqConfEvent_t (
      SystemStatus_t          systemStatus;
) CSTASysStatStatReqConfEvent_t

typedef enum SystemStatus_t {
   SS_INITIALIZING = 0,
   SS_ENABLED = 1,
   SS_NORMAL = 2,
   SS_MESSAGES_LOST = 3,
   SS_DISABLED = 4,
   SS_OVERLOAD_IMMINENT = 5,
   SS_OVERLOAD_REACHED = 6,
   SS_OVERLOAD_RELIEVED = 7
} SystemStatus_t;
```

Parameters

acsHandle
This is the handle for the opened ACS Stream.

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

eventType
This is a tag with the value **CSTA_SYS_STAT_REQ_CONF**, which identifies this message as an **CSTASysStatReqConfEvent.**

invokeID
This parameter specifies the requested instance of the function or event. It is used to match a specific function call request with its confirmation events.

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.

systemStatus
This parameter provides the application with a cause code defining the overall system status as fin Table 9-3

Table 9-4

Overall System Status Codestc "Overall System Status Cause Codes"\f t \l3§

Cause Code	Definition
Initializing	the driver/switch is re-initializing or restarting. This status indicates that the driver/switch is temporarily unable to respond to any requests. If provided, this status message shall be followed by an Enable status message to indicate that the initialization process is completed.
Enabled	request and responses are re-enabled, usually after a disruption or restart. This status indication shall be sent after an initializing status indicator has been sent and may be sent under other conditions. This status indicates that there are no outstanding monitor request.
Normal	this cause value can be sent at any time by the driver/switch to indicate that the status is normal. This status has no effect on other services.
Message Lost	this status indicates that a request and/or responses may have been lost, including Event Reports.
Disabled	this cause value indicates that existing monitor requests via **cstaMonitorStart**( ) have been disabled. Other requests and responses may also be disabled, but reject responses should be provided.
Overload Imminent	the driver/switch is about to reach a overload condition and the application should shed load to better the situation.

Overload Reached	the driver/switch has reach overload and may take initiative to shed load. This cause may be followed by action on the part of the driver/switch to decrease message traffic. This may include stopping existing or rejecting any new monitor requests sent by the client, and rejections to additional new service requests.
Overload Relieved	the driver/switch has determined that the overload condition has passed and normal application operation may continue.

Comments

This confirmation event provides the application with certain information regarding the state of the overall driver/switch system. This event is important for proper application operation and should be processed accordingly. This is especially important for cause values for the overload condition. If the driver/switch has informed the application that an overload condition is imminent all applications should attempt to decrease the overall traffic to the driver/switch. This can be accomplished, for example, by stopping all non-essential monitors on call or device objects on the switch thus reducing the traffic between the server and the switch. Frequent occurrence of the Overload Imminent cause value can be a symptoms of a poorly engineered system which should reviewed for proper loading.

cstaSysStatStart( )XE "cstaSysStatStart( )"§

This services allows the application to register for System Status event reporting. It can be used by an application to automatically receive a **CSTASysStatEvent** each time the status of the driver/switch changes.

Syntax

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

RetCode_t cstaSysStatStart (
      ACSHandle_t                acsHandle,
      InvokeID_t                 invokeID,
      SystemStatusFilter_t   statusFilter,
      PrivateData_t              *privateData),

typedef unsigned char   SystemStatusFilter_t;
#define           SF_INITIALIZING 0x80
#define           SF_ENABLED 0x40
#define           SF_NORMAL 0x20
#define           SF_MESSAGES_LOST 0x10
#define           SF_DISABLED 0x08
#define           SF_OVERLOAD_IMMINENT 0x04
#define           SF_OVERLOAD_REACHED 0x02
#define           SF_OVERLOAD_RELIEVED 0x01
```

Parameters

acsHandle
This is the handle to an active 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 valid 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.

statusFilter
This parameter is used to specify a filter for specific cause values in which the application is not interested. The parameter can be used by the application to filter out unwanted status information (e.g. the Normal status)

privateData
Private data extension mechanism. This is optional.

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 **CSTASysStatStartConfEvent** 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_NOCONN
This return value indicates that a previously active ACS Stream has been abnormally aborted.

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

Comments

This function is used to start a monitor for system status information. The system status information is provided via the **CSTASysStatEvent**. Only one System Status register is allowed per opened ACS Stream.

CSTASysStatStartConfEventXE "CSTASysStatStartConfEvent"§

This event is in response to the **cstaSysStatStart**( ) function and confirms an active System Status monitor. Once this event is issued the application will start to automatically receive unsolicited System Status events.

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_t;
                  union
                  {
                        CSTASysStatStartConfEvent  sysStatStart;
                  } u;
            } cstaConfirmation;
      } event;} CSTAEvent_t;
typedef struct CSTASysStatStartConfEvent_t (
      SystemStatusFilter_t          systemFilter;
) CSTASysStatStatStartConfEvent_t
```

Parameters

acsHandle
This is the handle for the opened ACS Stream.

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

eventType
This is a tag with the value **CSTA_SYS_STAT_START_CONF**, which identifies this message as an **CSTASysStatStartConfEvent.**

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.

statusFilter
This parameter is used to specify the filter type which is active on the System Status monitor requested by the application. The parameter identifies which filter was accepted by the driver/switch. Note that the filter returned by this function may be different than the filter requested in the **cstaSysStatStart**( ) service request. This can occur when the driver/switch rejected the request filter and selected a default 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 checked by the application to insure that the System Status monitor has been activated and that the requested filter is active.

cstaSysStatStop( )XE "cstaSysStatStop( )"§

This service is used to cancel a previously registered monitor for System Status information.

Syntax

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

RetCode_t cstaSysStatStop (
      ACSHandle_t                acsHandle,
      InvokeID_t                 invokeID,
      PrivateData_t          *privateData),
```

Parameters

acsHandle
This is the handle to an active 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 valid 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.

privateData
Private data extension mechanism. This is optional.

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 **CSTASysStatStopConfEvent** 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_NOCONN
This return value indicates that a previously active ACS Stream has been abnormally aborted.

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

Comments

This function is used to cancel a previously registered System Status monitor. Once a confirmation event is issued for this function, i.e. a **CSTASysStatStopConfEvent**, the driver/switch will

terminate automatic System Status event notification. If required, the application can still continue to poll for system status information using the **cstaSysStatReq**( ) service, even after a System Status register is closed.

CSTASysStatStopConfEventXE "CSTASysStatStopConfEvent"§

This event is in response to the **cstaSysStatStop**( ) function and confirms a cancellation of the active System Status monitor. Once this event is issued the application will not continue to receive unsolicited System Status events.

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

Parameters

acsHandle
This is the handle for the opened ACS Stream.

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

eventType
This is a tag with the value **CSTA_SYS_STAT_STOP_CONF**, which identifies this message as an CSTASysStatStopConfEvent.

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.

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 System Status monitor has been deactivated. Once this event is sent, automatic notification of System Status events will be discontinued. The application must poll using the **cstaSysStatReq**( ) service in order to obtain any System Status information.

cstaChangeSysStatFilter( )XE "cstaChangeSysStatFilter( )"§

This function is used to request a change in the filter options for automatic System Status event reporting for a specific ACS Stream. It allows the application to specify which System Status events it requires.

Syntax

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

RetCode_t cstaChangeSysStatFilter (
      ACSHandle_t                acsHandle,
      InvokeID_t                 *invokeID,
      SystemStatusFilter_t       statusFilter,
      PrivateData_t              *privateData),
```

Parameters

acsHandle
This is the handle to an active 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 valid 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.

statusFilter
This parameter identifies the new filter mask to be applied to the existing active System Status monitor. The new mask will replace the existing mask.

privateData
Private data extension mechanism. This is optional.

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 **CSTAChangeSysStatFilterConfEvent** 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_NOCONN
This return value indicates that a previously active ACS Stream has been abnormally aborted.

ACSERR_REQDENIED
This return value indicates that a ACS Stream is established but a requested capability has been

denied by the Client Library Software Driver.

Comments

This service is used whenever the application wishes to change a previously defined System Status event filter. Note that application will not receive any System Status message which has its bit mask turned off.

CSTAChangeSysStatFilterConfEventXE "CSTAChangeSysStatFilterConfEvent"§

This event occurs as a result of the **cstaChangeSysStatFilter**( ) service 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_t          acsHandle;EventClass_t     eventClass;       EventType_t
          eventType;
} ACSEventHeader_t;

typedef struct
{
      ACSEventHeader_t    eventHeader;
      union
      {           struct
            {                 InvokeID_t       invokeID_t;
                  union
                  {
      CSTAChangeSysStatFilterConfEvent  changeSysStatFilter;
                  } u;
            } cstaConfirmation;
      } event;} CSTAEvent_t;
typedef struct CSTAChangeSysStatFilterConfEvent_t (
      SystemStatusFilter_t   statusFilterSelected;
      SystemStatusFilter_t   statusFilterActive;
} CSTAChangeSysStatFilterConfEvent_t;
```

Parameters

acsHandle
This is the handle for the opened ACS Stream.

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

eventType
This is a tag with the value CSTA_CHANGE_SYS_STAT_FILTER_CONF , which identifies this message as an CSTAChangeSysStatFilterConfEvent.

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.

statusFilterSelected
This parameter specifies the System Status event filters which are active as a result of the **cstaChangeSysStatFilter**( ) service request. This filter may be different than the one requested by the application. This can occur if the implementation rejects a particular filter request.

eventFilterActive
This parameter indicates the filters which are already active on the given CSTA association.

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 insure that the event filter requested has been activated and which filters are already active on the given System Status monitor.

CSTASysStatEventXE "CSTASysStatEvent"§

This unsolicited event informs the application of the overall system status of the driver/switch. The application must register for System Status events before this event is sent to the application.

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
                  {
                        CSTASysStatEvent_t  sysStat;
                  } u;
            } cstaEventReport;
      } event;} CSTAEvent_t;
typedef struct
{
      SystemStatus_t          systemStatus;
} CSTASysStatEvent_t;

typedef enum SystemStatus_t {       SS_INITIALIZING = 0,       SS_ENABLED = 1,
      SS_NORMAL = 2,       SS_MESSAGES_LOST = 3,       SS_DISABLED = 4,
      SS_OVERLOAD_IMMINENT = 5,          SS_OVERLOAD_REACHED = 6,
      SS_OVERLOAD_RELIEVED = 7} SystemStatus_t;
```

Parameters

acsHandle
This is the handle for the opened ACS Stream.

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_SYS_STAT**, which identifies this message as an **CSTASysStatEvent.**

monitorCrossRefID
This parameter is unused in this message..

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.

systemStatus
This parameter provides the application with a cause code defining the overall system status. See Table 9-5 for the possible values of this parameter.

Comments

This event provides the application with certain information regarding the state of the overall driver/switch system. This event is important for proper application operation and should be processed accordingly. This is especially important for cause values for the overload condition. If the driver/switch has informed the application that an overload condition is imminent all applications should attempt to decrease the overall traffic to the driver/switch. This can be accomplished, for example, by stopping all non-essential monitors on call or device objects on the switch thus reducing the traffic between the server and the switch. Frequent occurrence of the Overload Imminent event can be a symptoms of a poorly engineered system which should reviewed for proper loading.

Certain, non-essential cause values can be sent at any time or depending on the driver/switch implementation even at regular intervals (e.g. the Normal cause value) to indicate that the system status is O.K. and operating normally. This can be turned off by the application to avoid the overhead associated with processing these normal messages. This is accomplished by changing the event filter type by using the **cstaChangeSysStatFilter**( ) service. This service can be used to discontinue the delivery of "non-essential" system status events to the application.

cstaSysStatEndedEventXE "cstaSysStatEndedEvent"§

This service is used by the driver to cancel a previously registered monitor for System Status information.

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
                        {
                                CSTASysStatEndedEvent_t  sysStatEnded;
                        } u;
                } cstaEventReport;
        } event;} CSTAEvent_t;
typedef struct CSTASysStatEndedEvent_t {
   Nulltype       null;
} CSTASysStatEndedEvent_t;
```

Parameters

acsHandle
This is the handle for the opened ACS Stream.

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_SYS_STAT_ENDED**, which identifies this message as an CSTASysStatStopEvent.

monitorCrossRefID
This parameter is unused in this message.

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.

System Status : Driver/Switch as the Client

XE "System status:Driver/switch as the client"§This section defines the services which provide system level status information to the driver/switch form the application. The System Status service is bi-directional and thus the client/server relationship (see Figure 9-2) can be reversed.

CSTASysStatReqEventXE "CSTASysStatReqEvent"§

This unsolicited event is sent by the driver/switch to request system status information from the application.

Syntax

The following structure shows only the relevant portions of the unions for this message. See section ***4.3 ACS Data Types*** and ***4.6 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
                  {                 InvokeID_t       invokeID;
                  union
                  {
                        CSTASysStatReqEvent_t    sysStatRequest;
                  } u;
            } cstaRequestEvent;
      } event;} CSTAEvent_t;
typedef struct CSTASysStatReqEvent_t {
   Nulltype       null;
} CSTASysStatReqEvent_t;
```

Parameters

acsHandle
This is the handle for the opened 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_SYS_STAT_REQ**, which identifies this message as an **CSTASysStatReqEvent.**

InvokeID
This parameter identifies the instance of the request generated by the switch/driver. This same value must be used, unchanged, in the response to this event (**cstaSysStatReqConf**( )).

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 sent by the driver/switch to request status information pertaining to the application. It is the bi-directional equivalent of the **cstaSysStatReq**( ) function which is issued by the application to request status information from the driver/switch. The application responds to this unsolicited event request utilizing the **cstaSysStatReqConf**( ) function.

cstaSysStatReqConf( )XE "cstaSysStatReqConf( )"§

This service is used to respond to a **CSTASysStatReqEvent** unsolicited event from the driver/switch. It provides the driver/switch with information regarding the status of the application.

Syntax

```
#include <csta.h>

RetCode_t cstaSysStatReqConf (

      ACSHandle_t          acsHandle,
      InvokeID_t           *invokeID,
      SystemStatus_t       systemStatus,
      PrivateData_t        *privateData);

typedef enum SystemStatus_t {  SS_INITIALIZING = 0,  SS_ENABLED = 1,  SS_NORMAL
      = 2,  SS_MESSAGES_LOST = 3,  SS_DISABLED = 4,  SS_OVERLOAD_IMMINENT
      = 5,      SS_OVERLOAD_REACHED  =  6,      SS_OVERLOAD_RELIEVED  =  7}
      SystemStatus_t;
```

Parameters

acsHandle
This is the handle to an active ACS Stream.

InvokeID
The value of this parameter must be the same (unchanged) as that provided in the **cstaSysStatReqEvent** so that the driver/switch can match an instance of a service request with the response to that request.

systemStatus
This parameter provides the driver/switch with a cause code defining the overall system status. See Table 9-6 for the possible values of this parameter.

privateData
Private data extension mechanism. This is optional.

Return Values

This function never returns an invoke identifier since there is no confirmation event for this service. The function does return errors conditions during the processing of the request by the API Client Library. A return value of zero (0) indicates that the request has been accepted by the Library.

Possible local error returns are (negative returns):

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

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

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

Comments

This confirmation response provides the driver/switch with certain information regarding the state of the overall application. The information can be used by the driver/switch to determine the overall state of the application. The driver/switch may act on this information in order to insure proper end-to-end system operation and performance. Frequent occurrence of the Overload Imminent cause value can be a symptoms of a poorly engineered application system which should re-

viewed for proper loading.

cstaSysStatEventSend( )XE "cstaSysStatEventSend( )"§

This service is used to send application system status information in the form of an unsolicited event to the driver/switch without a formal request for the information. This status information can be sent at any time.

Syntax

```
#include <csta.h>

RetCode_t cstaSysStatEvent (
      ACSHandle_t              acsHandle,
      SystemStatus_t           systemStatus,
      PrivateData_t            *privateData);

typedef enum SystemStatus_t {   SS_INITIALIZING = 0,   SS_ENABLED = 1,   SS_NORMAL
      = 2,   SS_MESSAGES_LOST = 3,   SS_DISABLED = 4,   SS_OVERLOAD_IMMINENT
      = 5,       SS_OVERLOAD_REACHED  =  6,       SS_OVERLOAD_RELIEVED  =  7}
      SystemStatus_t;
```

Parameters

acsHandle
This is the handle to an active ACS Stream.

systemStatus
This parameter provides the driver/switch with a cause code defining the overall system status. See Table 9-7 for the possible values of this parameter.

privateData
Private data extension mechanism. This is optional.

Return Values

This function never returns an invoke identifier since there is no confirmation event for this service. The function does return errors conditions during the processing of the request by the API Client Library. A return value of zero (0) indicates that the request has

been accepted by the Library.

Possible local error returns are (negative returns):

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

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

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

Comments

This unsolicited service event is sent to the driver/switch in order to inform it of the state of the overall application system. The driver/switch may act on this information in order to insure proper end-to-end system operation and performance. Frequent occurrence of the Overload Imminent cause value can be a symptoms of a poorly engineered application system which should reviewed for proper loading.