A MITEL
PRODUCT
GUIDE

MiContact Center Enterprise

Open Application Server (OAS) - API Programmer's Guide

Release 9.6
Document Version 1.0

September 2022

Mitel
Powering connections

Notices

The information contained in this document is believed to be accurate in all respects but is not warranted by **Mitel Networks™ Corporation (MITEL®).**

The information is subject to change without notice and should not be construed in any way as a commitment by Mitel or any of its affiliates or subsidiaries. Mitel and its affiliates and subsidiaries assume no responsibility for any errors or omissions in this document. Revisions of this document or new editions of it may be issued to incorporate such changes. No part of this document can be reproduced or transmitted in any form or by any means - electronic or mechanical - for any purpose without written permission from Mitel Networks Corporation.

Trademarks

The trademarks, service marks, logos and graphics (collectively “Trademarks”) appearing on Mitel's Internet sites or in its publications are registered and unregistered trademarks of Mitel Networks Corporation (MNC) or its subsidiaries (collectively "Mitel") or others. Use of the Trademarks is prohibited without the express consent from Mitel. Please contact our legal department at legal@mitel.com for additional information. For a list of the worldwide Mitel Networks Corporation registered trademarks, please refer to the website: http://www.mitel.com/trademarks.

®,™ Trademark of Mitel Networks Corporation

ABSTRACT

The Open Application Server (OAS) is an open, scalable platform on which Computer Telephony Integration (CTI) media applications can be based. An easy to use call and media control Application Programming Interface (API) enables applications to share network resources while still reserving resources for mission critical media applications. The API is based on the Novell NetWare Telephony Services API (TSAPI) and contains additional functions that enhance the API with functionality not included in the standard TSAPI (for example, media services). For general information about the API, please refer to *NetWare® Telephony Services Release 2 - Telephony Services Application Programming Interface (TSAPI)*.

The *Open Application Server API Programmer's Guide* describes the ETP extensions to TSAPI provided by the OAS. In addition to the TSAPI programming interface, which incorporates the following standards:

- ECMA (European Computer Manufacturers Association. ECMA is an international, European-based industry association that focuses on the standardization of information and communications systems.)
- CSTA (Computer-Supported Telecommunications Applications)

and the following services: Telephony Call Control Services

- Call/Device Monitoring Services
- Query Services
- the OAS API incorporates the following services: Media Control Services
- Virtual Device Services
- Inter-application Communication Device Services

HOW THIS GUIDE IS ORGANIZED

This guide is structured following the same principles as the TSAPI specification. For the reader who is familiar with that specification, this simplifies finding information. The chapters and appendices are:

- "Introduction" - Discusses the Open Application Server—its TSAPI foundation and the OAS API product architecture.
- "Functional Call Model" - References the functional call model and terminology specific to the OAS API.
- "Control Services" - References the control services that the OAS supports.
- "Switching Function Services" - Discusses the OAS's Switching Function Services available to control calls.
- "Status Reporting Services" - Specifies the status reporting functions and confirmation events that differ from those specified by TSAPI. Describes OAS-specific unsolicited event messages coming from the OAS system.

- "OAS Data Types" - References the data types used by the functions and messages defined for the OAS API.
- "References"
- Appendix A, "TSAPI Services Supported" - Lists the TSAPI services and whether the OAS supports them.
- Appendix B, "Universal Failure Events" – Describes the CSTA and ETP Universal Failure Events.
- Appendix C, "ETP CSTA Private Data" - Discusses the private data mechanism, which extends the standard features of CSTA to allow OAS applications to invoke OAS-specific features.

CONVENTIONS

SERVICE FUNCTIONS VERSUS EVENTS

All OAS service functions start with **etp** and all OAS confirmation and unsolicited events start with **ETP**. For example:

etpAllocateResources( ) (service function)

ETPAllocateResourcesConfEvent (confirmation event)

USE OF EMPHASIS

The conventions used in this reference to distinguish service function names, parameter names, parameter values, message events, etc. are as follows:

This font is used for service function names and parameter values. For example, service function names include **etpPlay( )** and **etpCollectDigits( )**. Sample parameter values are **ETPCONFIRMATION** and **ETP_MAX_DIGITS**.

This font is used for parameter names. For example: *acsHandle* and *invokeID*.

This font is used for message events. For example: **ETPPlayConfEvent**.

This font is used for error messages. For example: *ACSERR_BADHDL*.

`This font` is used for programming statements and declarations. For example:

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

RetCode_t etpPlay (
        ACSHandle_t      acsHandle,
        InvokeID_t       invokeID,
        ConnectionID_t   *call,
                           int             messageID,
        ETPPlayList_t     *playList);
```

COMPANION REFERENCES

The *Open Application Server API Programmer's Guide* points to other references for additional information. When using this guide, you will also need the following:

- ApplicationLink, Application Programmer's Guide. EN/LZT 1022273
- Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI). Novell, May 1995.

INTRODUCTION

THE TELEPHONY SERVICES API (TSAPI) FOUNDATION

The Telephony Services API (TSAPI) is a set of APIs that allows applications to monitor and control devices in a switching domain to bring together the two most common pieces of equipment on an end user's desktop—the telephone and computer. It supports telephony control capabilities in a generic, switch independent way (e.g., supports PBXs from various vendors).

TSAPI is based on international standards for Computer Telephony Integration (CTI) telephony services. Specifically, the ECMA CTI standard definition of Computer- Supported Telecommunications Applications (CSTA Phase 1) is the foundation for TSAPI. ECMA is an international, European-based industry association that focuses on the standardization of information and communications systems. The CSTA standard is a technical agreement reached by an open, multi-vendor consortium of major switch and computer vendors. Since CSTA services and protocol definitions are the basis for TSAPI, TSAPI provides a generic, switch-independent API. The TSAPI programming interface definition incorporates ECMA CSTA telephony call control services, call/device monitoring services, and query services.

Open Application Server (OAS) builds on the TSAPI foundation, extending its capabilities with media services, virtual device functions, and other functionality. It allows applications to utilize resources in a MX-ONE network in a cost efficient yet flexible way.

PURPOSE

This reference specifies the OAS APIs (i.e., the additions to TSAPI that provide the extended functionality of the Open Application Server). The API is specified in terms of its services and C programming language syntax. OAS API supports Microsoft Windows 7 and 10, Microsoft Windows Server 2K12, 2K16 and 2K19.

For information regarding generic TSAPI services and syntax, please refer to Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI).

FUNCTIONAL CALL MODEL

The Open Application Server API functional call model is the same as that for TSAPI. For details, refer to *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*.

CONTROL SERVICES

The Open Application Server supports the control services specified in Section 4, "Control Services," of the *Netware® Telephony Services Release 2 - Telephony Services Application*

Programming Interface (TSAPI).

SWITCHING FUNCTION SERVICES

Applications use the Switching Function Services available through the Open Application Server API to control calls. For OAS, the Dialed Number Identification Service (DNIS) name will be displayed for an ACD call when the dialed number value maps to a defined name; if there is no defined name, the DNIS number will be provided.

This chapter discusses the following types of Switching Function Services:

- Physical and Logical Device Call Control Services
- Virtual Device Call Control Services
- Inter-application Communication Device Control Services
- Media Control Services

PHYSICAL AND LOGICAL DEVICE CALL CONTROL SERVICES

A physical device is one associated with a *real* telephone extension, whereas a logical device is a software-defined device (within the switch) and is associated with an extension. Logical devices include ACD Groups and CTI Groups.

The call control services supported by OAS for physical and logical devices can be divided into three categories:

- Standard TSAPI services. These are a subset of TSAPI Release 2 supported by the ApplicationLink product. See the *ApplicationLink Application Programmer's Guide* for details.
- Standard TSAPI services extended in OAS to provide enhanced functions. The extensions are implemented by employing the private data mechanism provided in TSAPI. The extensions are detailed in this section.
- OAS-specific services to provide enhanced functions. These services are detailed in this section.

cstaDeflectCall( )

The **cstaDeflectCall( )** service redirects an Alerting or Connected call to a given number. When the call is alerting at either an ACD or CTI group, the **cstaDeflectCall( )** service is extended using the private data mechanism to accept a retain position flag. On the **cstaDeflectCall( )** service API call, the **ETP_PD_RetainPositionFlag** private data can be passed to the OAS client library. Presence of the private data on the **cstaDeflectCall( )** signals OAS to deflect the call with the call's position in the queue retained. Absence of this parameter means that the switch does not retain the position of the call in the queue.

For more information, refer to Appendix C, "ETP CSTA Private Data," ApplicationLink Application Programmer's Guide, and Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI).

Syntax

The following structure shows only the relevant additional portions of the unions for this message. For a complete description of the event structure, refer to Appendix C, “ETP CSTA Private Data,” Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

```
typedef struct ETP_PD_RetainPositionFlag_t  {
         char              vendor[32];
         unsigned short length;
} ETP_PD_RetainPositionFlag_t;
```

Parameters

vendor

Stores the manufacturer object identifier. For the **ETP_PD_RetainPositionFlag**, the sequence is:

{0x2B, 0x0C, 0x02, 0x89, 0x3D, 0x28, 0x04, 0x0}

length

Length of the data portion of the private data structure that follows the *length* parameter. Since there is no data portion, *length* must contain **0** (zero).

Comments

If the function returns an error with a cost code service busy, another service is trying to deflect to the same destination. Retry; the request should be successful.

Note the following concerning the structure and parameters:

- It is assumed that the byte alignment for the structure is 1. In other words, there is no gap between the *vendor* parameter and the *length* parameter.
- There is no data parameter.
- The retain position flag is valid only for calls on a CTI or ACD group, and when the ‘deflected to’ device is an analog, CTI or ACD device.

cstaMonitorDevice ( )

The cstaMonitorDevice( ) service monitors the requested Device. On the cstaMonitorDevice() service API call, the ETPTenantID_t is added to private data and can be passed to the OAS client library.Presence of the private data on the cstaMonitorDevice () signals OAS to search the device in the specified Tenant list.For more information, refer to Appendix C, "ETP CSTA Private Data,"ApplicationLink Application Programmer's Guide, and *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI).*

Syntax

The following structure shows only the relevant additional portions of the unions for his message. For a complete description of the event structure, refer to Appendix C, “ETP CSTA Private Data,” Chapter 7, “OAS Data Types,” and the Netware® Telephone Services™ Release 2 - Telephony

Services Application Program Interface (TSAPI), the ACS Data Types and CSTA Data Types sections.

```
typedef struct ETPTenantID_t {
                TenantID_t      TenantID;
} ETPTenantID_t;
```

Parameters for Private data

vendor

Stores the manufacturer object identifier. For the **ETPTenantID**, the sequence is:

{0x2B, 0x0C, 0x02, 0x89, 0x3D, 0x28, 0x05, 0x0}

length

Length of the data portion of the private data structure that follows the length parameter. Since there is no data portion, length must contain 0 (zero).

Comments

If the function returns an error with a cause code as security violation, wrong tenant id is sent in request.

Note the following concerning the structure and parameters:· It is assumed that the byte alignment for the structure is 1. In other words, there is no gap between the vendor parameter and the length parameter.

Sample code to fill in Tenant ID in private data:

```
int dataLen = sizeof(PrivateData_t)+sizeof(ETPTenantID_t);
ETPTenantID_t  etpTID;
PrivateData_t  *privData  = (PrivateData_t*)malloc(dataLen);
memset((void*)&etpTID,0,sizeof(ETPTenantID_t));
strncpy(etpTID.TenantID,  (LPCSTR)generalPage.m_tenantID,63);
memset((void*)privData,  0,  dataLen);
memcpy((void*)privData->vendor,TenantVendorID,8); privData-
>length  = sizeof(ETPTenantID_t)+1; memcpy((void*)privData-
>data,  &etpTID,  sizeof(ETPTenantID_t));
ret = MonitorDevice(handle,  invokeID,  &device,  &filter,  privData);
```

etpAssociateData( )

The **etpAssociateData( )** service associates/attaches data to a call so that subsequent unsolicited events relating to the call contain the data.

Syntax

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

RetCode_t etpAssociateData  (
         ACSHandle_t               acsHandle,
         InvokeID_t                invokeID,
         ConnectionID_t            *call,
         ETPAssociatedData_t       *associatedData);
```

Parameters

acsHandle

The handle for 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 identifiers in the **acsOpenStream( )**. The parameter is ignored by the ACS Library when the Stream is set for library-generated identifiers.

call

A pointer to the connection identifier of the call to which resources are to be allocated.

The structure for the **ConnectionID_t** is

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

Parameters

callID

Unique id used for the call.

deviceID

device identifier of the device.

devIDType

The four byte enum which specifies the type of device either STATIC or DYNAMIC.

associatedData

A pointer to the data to be associated with the call.

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 **ETPAssociatedDataConfEvent** message to ensure that the service request has been acknowledged and processed by the server.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
A bad or unknown *acsHandle* was provided by the application.

ACSERR_STREAM_FAILED
A previously active ACS Stream has been abnormally aborted.

Comments

This service is valid for virtual devices as well as physical and logical devices. The unsolicited events that support associated data are:

- CSTAConferencedEvent
- CSTAConnectionClearedEvent
- CSTADeliveredEvent
- CSTADivertedEvent
- CSTAEstablishedEvent
- CSTAQueuedEvent
- CSTATransferredEvent

ETPAssociateDataConfEvent

ETPAssociateDataConfEvent provides a positive response from the server for a previous **etpAssociateData( )** request.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

```
typedef struct
      {
      ACSEventHeader_t       eventHeader;
      Union
          {
          Struct
                  {
                  InvokeID_t invokeID;
                          union
                          {
                          ETPAssociateDataConfEvent_t assocData;
                          } u;
                      } etpConfirmation;
          } event;
          } ETPEvent_t;
typedef struct ETPAssociateDataConfEvent_t  {
       Nulltype       null;
} ETPAssociateDataConfEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPCONFIRMATION**, which identifies this message as an ETP confirmation event.

eventType

Tag value **ETP_ASSOCIATE_DATA_CONF**, which identifies this message as an **ETPAssociateDataConfEvent**.

invokeID

Specifies the function service request instance for the service, which was processed at the server or switch. This identifier is provided to the application when a service request is made.

etpCancelCallback( )

The **etpCancelCallback( )** service allows an application to cancel a previously set callback on a device. A specific callback can be canceled by including the called party device ID. If the called party device ID is not included, then all callbacks on the device are removed.

Syntax

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

RetCode_t etpCancelCallback  (
                  ACSHandle_t     acsHandle,
                  InvokeID_t      invokeID,
                  DeviceID_t      *deviceID,
                  DeviceID_t      *callbackToCancel);
```

Parameters

acsHandle

The handle for 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 identifiers in the **acsOpenStream( )**. The parameter is ignored by the ACS Library when the Stream is set for library-generated identifiers.

deviceID

A pointer to the device identifier of the device which originates the **etpCancelCallback( )** service.

callbackToCancel

A pointer to the device identifier of the called party device. If the device identifier is not included (i.e., the value of *CallBackToCancel* is null), all callbacks on the device are removed.

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 **ETPCancelCallbackConfEvent** message to ensure that the service request has been acknowledged and processed by the server and switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
A bad or unknown *acsHandle* was provided by the application.

ACSERR_STREAM_FAILED
A previously active ACS Stream has been abnormally aborted.

Comments

This service is only valid for a physical device in the Idle state.

ETPCancelCallbackConfEvent

ETPCancelCallbackConfEvent provides a positive response from the server for a previous **etpCancelCallback( )** request.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct
        {
        ACSEventHeader_t        eventHeader;
        union {
                  struct {
                          InvokeID_t      invokeID;
                            union {
                                ETPCancelCallbackConfEvent_t    cancelCB;
                                    } u;
                           } etpConfirmation;
                  } event;
  } ETPEvent_t;
        typedef struct ETPCancelCallbackConfEvent_t  {
        Nulltype        null;
        } ETPCancelCallbackConfEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPCONFIRMATION**, which identifies this message as an ETP confirmation event.

eventType

Tag value **ETP_CANCEL_CALLBACK_CONF**, which identifies this message as an **ETPCancelCallbackConfEvent**.

invokeID

Specifies the function service request instance for the service that was processed at the server or switch. This identifier is provided to the application when a service request is made.

etpDeflectCallWithData( )

This function allows an application to deflect a call with user associated data. The user associated data will be provided to the device to which the call is diverted in CSTADeliveredEvent.

Syntax

```
#include <acs.h>
#include <csta.h>
#include <etp.h>
RetCode_t etpDeflectCallWithData  ( ACSHandle_t acsHandle,
     InvokeID_t                     invokeID,
     ConnectionID_t                 *deflectCall,
     DeviceID_t                     *calledDevice
     ETPAssociatedData_t            *associatedData,
     PrivateData_t                  *privateData);
```

Parameters

acsHandle

The handle for 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 identifiers in the **acsOpenStream( )**. The parameter is ignored by the ACS Library when the Stream is set for library-generated identifiers.

Deflectcall

A pointer to the connection identifier of the call to be deflected.

calledDevice

A pointer to the device identifier of device to which the call is to be deflected.

associatedData

A pointer to the data to be associated with the call.

privateData

Indicates the privateData.

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 **ETPDeflectCallWithDataConfEvent** message to ensure that the service request has been acknowledged and processed by the server and switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
A bad or unknown *acsHandle* was provided by the application.

ACSERR_STREAM_FAILED
A previously active ACS Stream has been abnormally aborted.

Comments

This service is only valid for calls on a virtual device.

Application can use this functionality to keep track of a call when it is deflected to another MX-ONE node. To see this associated data, the deflected destination must be monitored by another OAS or ApplicationLink.

ETPDeflectCallWithDataConfEvent

ETPDeflectCallWithDataConfEvent provides a positive response from the server for a previous **etpDeflectCallWithData( )** request.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

```
typedef struct {
        ACSEventHeader_t       eventHeader;
                union {
                    struct {
                             InvokeID_t     invokeID;
                             union {
                             ETPDeflectCallWithDataConfEvent_t      assocData;
                                        } u;
                             } etpConfirmation;
                    } event;
} ETPEvent_t;

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

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPCONFIRMATION**, which identifies this message as an ETP confirmation event.

eventType

Tag value ETP_DEFLECT_CALL_WITH_DATA_CONF, which identifies this message as an ETPDeflectCallWithDataConfEvent.

invokeID

Specifies the function service request instance for the service, which was processed at the server or switch. This identifier is provided to the application when a service request is made.

etpEnterAccountCode( )

The **etpEnterAccountCode( )** service associates an account code with either an active call in speech or a new call that it starts. If the call is currently active (the device is in the Connected state), the account code uses another line that appears on the telephone set. If the device is in the Idle state, a line appearance will be selected, the account code dialed, and the line remains in the Dial Tone state ready for manual dial or dial through a **cstaMakeCall( )** service request.

Syntax

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

RetCode_t etpEnterAccountCode  ( ACSHandle_t
                                 acsHandle,
                InvokeID_t      invokeID,
                ConnectionID_t *call,
                AccountCode_t   accountCode);
```

Parameters

acsHandle

The handle for 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 identifiers in the **acsOpenStream( )**. The parameter is ignored by the ACS Library when the Stream is set for library-generated identifiers.

call

A pointer to the connection identifier of the call to which the account code is associated. Used when the device is in the Connected state.

accountCode

A null terminated character string that contains the account code to be entered. The code is market and system specific, and the maximum number of characters is 10.

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 **ETPEnterAccountCodeConfEvent** message to ensure that the service request has been acknowledged and processed by the server and switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
A bad or unknown *acsHandle* was provided by the application.

ACSERR_STREAM_FAILED
A previously active ACS Stream has been abnormally aborted.

Comments

This service is only valid for a physical device in the Idle or Connected state.

ETPEnterAccountCodeConfEvent

ETPEnterAccountCodeConfEvent provides a positive response from the server for a previous **etpEnterAccountCode( )** request.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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


typedef struct {
        ACSEventHeader_t         eventHeader;
                  union {
                 struct {
                           InvokeID_t     invokeID;
                                  union {
                                        ETPEnterAccountCodeConfEvent_t

                                   enterAccountCode;
                                         } u;
                           } etpConfirmation;
                } event;
                } ETPEvent_t;
typedef struct ETPEnterAccountCodeConfEvent_t  {
        Nulltype          null;
} ETPEnterAccountCodeConfEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPCONFIRMATION**, which identifies this message as an ETP confirmation event.

eventType

Tag value ETP_ENTER_ACCOUNT_CODE_CONF, which identifies this message as an ETPEnterAccountCodeConfEvent.

invokeID

Specifies the function service request instance for the service that was processed at the server or switch. This identifier is provided to the application when a service request is made.

etpEnterAuthorizationCode( )

The **etpEnterAuthorizationCode( )** service allows an application to dial an authorization code on an idle device.

Syntax

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

RetCode_t etpEnterAuthorizationCode  (
        ACSHandle_t            acsHandle,
        InvokeID_t             invokeID,
        DeviceID_t             *deviceID,
        AccountCode_t          authCode);
```

Parameters

acsHandle

The handle for 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 identifiers in the **acsOpenStream( )**. The parameter is ignored by the ACS Library when the Stream is set for library-generated identifiers.

deviceID

A pointer to the device identifier of the device with which an authorization code will be associated.

authCode

A null terminated character string that contains the authorization code to be entered. The code is market and system specific, and the maximum number of characters is 10.

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 **ETPEnterAuthorizationCodeConfEvent** message to ensure that the service request has been acknowledged and processed by the server and switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
A bad or unknown *acsHandle* was provided by the application.

ACSERR_STREAM_FAILED
A previously active ACS Stream has been abnormally aborted.

Comments

This service is only valid for a physical device in the Idle state.

ETPEnterAuthorizationCodeConfEvent

ETPEnterAuthorizationCodeConfEvent provides a positive response from the server for a previous **etpEnterAuthorizationCode( )** request.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
        ACSEventHeader_t        eventHeader;
                          union {
                          struct {
                                  InvokeID_t  invokeID;
                                  union {
                        ETPEnterAuthorizationCodeConfEvent_t  enterAuthCode;
                                        } u;
                                 } etpConfirmation;
                           } event;
        } ETPEvent_t;

typedef struct ETPEnterAuthorizationCodeConfEvent _t {
        Nulltype        null;
        } ETPEnterAuthorizationCodeConfEvent _t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPCONFIRMATION**, which identifies this message as an ETP confirmation event.

eventType

Tag value ETP_ENTER_AUTHORIZATION_CODE_CONF, which identifies this message as an ETPEnterAuthorizationCodeConfEvent.

invokeID

Specifies the function service request instance for the service that was processed at the server or switch. This identifier is provided to the application when a service request is made.

etpMessageDiversion( )

The **etpMessageDiversion( )** service allows an application to set or cancel diversion of a message to a predefined destination for specified reasons when the device is in the Idle state.

Syntax

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

RetCode_t       etpMessageDiversion  ( ACSHandle_t
                acsHandle, InvokeID_t
                invokeID, DeviceID_t
                *deviceID, Boolean
                divertMessage, Int
                diversionType, TimeOrDate_t
                timeOrDate);
```

Parameters

acsHandle

The handle for 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 identifiers in the **acsOpenStream( )**. The parameter is ignored by the ACS Library when the Stream is set for library-generated identifiers.

deviceID

A pointer to the device identifier of the device to which diversion to a predefined destination is to be set or canceled.

divertMessage

If set to **true**, diversion to a predefined destination is set. If set to **false**, message diversion is canceled.

diversionType

Specifies the reason for the diversion. Values are between **0** and **9**, and are system and market specific.

timeOrDate

A null terminated 4-byte string that can contain date or time depending upon the reason specified in *diversionType*. For example, if the reason were “Out for Jury Duty,” then the string would contain the date; if the reason were “Out for Lunch,” then it would contain the time. The string is system and market specific.

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 **ETPMessageDiversionConfEvent** message to ensure that the service request has been acknowledged and processed by the server and switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
A bad or unknown *acsHandle* was provided by the application.

ACSERR_STREAM_FAILED
A previously active ACS Stream has been abnormally aborted.

Comments

This service is only valid for a physical device in the Idle state.

ETPMessageDiversionConfEvent

ETPMessageDiversionConfEvent provides a positive response from the server for a previous **etpMessageDiversion( )** request.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

```
typedef struct {
               ACSHandle_t            acsHandle;
               EventClass_t           eventClass;
               EventType_t            eventType;
} ACSEventHeader_t;
typedef struct {
           ACSEventHeader_t     eventHeader;
              union {
                 struct {
                         InvokeID_t      invokeID;
                         union {
                               ETPMessageDiversionConfEvent_t      msgDiversion;
                               } u;
           } etpConfirmation;
        } event;
} ETPEvent_t;
typedef struct ETPMessageDiversionConfEvent_t  {
               Nulltype       null;
} ETPMessageDiversionConfEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPCONFIRMATION**, which identifies this message as an ETP confirmation event.

eventType

Tag value ETP_MESSAGE_DIVERSION_CONF, which identifies this message as an TPMessageDiversionConfEvent.

invokeID

Specifies the function service request instance for the service that was processed at the server or switch. This identifier is provided to the application when a service request is made.

etpPressProgrammableKey( )

The **etpPressProgrammableKey( )** service allows an application to press any programmable key on the device by identifying the key number as defined by the MX-ONE telephone set key numbering system.

Syntax

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

RetCode_t etpPressProgrammableKey  (
                ACSHandle_t            acsHandle,
                InvokeID_t             invokeID,
                DeviceID_t             *deviceID,
                KeyNumber_t            keyNumber);
```

Parameters

acsHandle

The handle for 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 identifiers in the **acsOpenStream( )**. The parameter is ignored by the ACS Library when the Stream is set for library-generated identifiers.

deviceID

A pointer to the device identifier of the device on which the application will press the programmable key.

keyNumber

The key to be pressed as defined by the MX-ONE telephone set key numbering system. Any key can be pressed except digits, the transfer key, the conference key, and the clear key. Use caution when invoking this service—the key numbering on different types of digital telephone sets on the MX-ONE is not the same.

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 **ETPPressProgrammableKeyConfEvent** message to ensure that the service request has been acknowledged and processed by the server and switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL

A bad or unknown *acsHandle* was provided by the application.

ACSERR_STREAM_FAILED

A previously active ACS Stream has been abnormally aborted.

Comments

This service is only valid for a physical device in the Idle state.

ETPPressProgrammableKeyConfEvent

ETPPressProgrammableKeyConfEvent provides a positive response from the server for a previous **etpPressProgrammableKey( )** request.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
     ACSEventHeader_t         eventHeader;
               union {
                struct {
                          InvokeID_t    invokeID;
                               union {
               ETPPressProgrammableKeyConfEvent_t       pressProgKey;
                                     } u;
                          } etpConfirmation;
               } event;
} ETPEvent_t;

typedef struct ETPPressProgrammableKeyConfEvent _t {
```

```
                Nulltype        null;
} ETPPressProgrammableKeyConfEvent _t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPCONFIRMATION**, which identifies this message as an ETP confirmation event.

eventType

Tag value ETP_PRESS_PROGRAMMABLE_KEY_CONF, which identifies this message as an ETPPressProgrammableKeyConfEvent.

invokeID

Specifies the function service request instance for the service that was processed at the server or switch. This identifier is provided to the application when a service request is made.

etpSendDTMF( )

The **etpSendDTMF( )** service transmits a series of DTMF signals on an established call in the Connected state.

Note: The etpSendDTMF( ) service is a Media Control Service as well as a Physical and Logical Device Call Control Service

Syntax

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

RetCode_t etpSendDTMF (
        ACSHandle_t             acsHandle,
        InvokeID_t              invokeID,
        ConnectionID_t          *call,
        DTMFList_t              dtmfList);
```

Parameters

acsHandle

The handle for 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 identifiers in the **acsOpenStream( )**. The parameter is ignored by the ACS Library when the Stream is set for library-generated identifiers.

call

A pointer to the connection identifier of the call to which the DTMF signals are to be sent.

dtmfList

A null terminated string containing the list of DTMF digits to be sent. Valid values are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, *, #. The string can contain from 1 to 20 characters.

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 **ETPSendDTMFConfEvent** message to ensure that the service request has been acknowledged and processed by the server and switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
A bad or unknown *acsHandle* was provided by the application.

ACSERR_STREAM_FAILED
A previously active ACS Stream has been abnormally aborted.

Comments

This service is only valid for a physical or virtual device.

ETPSendDTMFConfEvent

ETPSendDTMFConfEvent provides a positive response from the server for a previous **etpSendDTMF( )** request.

Note: The etpSendDTMF( ) service is a Media Control Service as well as a Physical and Logical Device Call Control Service.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
        ACSEventHeader_t        eventHeader;
                union {
                          struct {
                               InvokeID_t      invokeID;
                                  union {
                                          ETPSendDTMFConfEvent_t     sendDTMF;
                                          } u;
                                    } etpConfirmation;
                                } event;
} ETPEvent_t;
  typedef struct ETPSendDTMFConfEvent_t  {
              Nulltype   null;
} ETPSendDTMFConfEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPCONFIRMATION**, which identifies this message as an ETP confirmation event.

eventType

Tag value **ETP_SEND_DTMF_CONF**, which identifies this message as an **ETPSendDTMFConfEvent**.

invokeID

Specifies the function service request instance for the service that was processed at the server or switch. This identifier is provided to the application when a service request is made.

etpSetACDGroupForward( )

The **etpSetACDGroupForward( )** service allows an application that is monitoring an Automatic Call Distribution (ACD) supervisor to request to forward an ACD group to another destination.

Syntax

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

RetCode_t etpSetACDGroupForward  ( ACSHandle_t
                                   acsHandle,
        InvokeID_t      invokeID,
        DeviceID_t      *deviceID,
        Boolean         forwardACDGroup,
        DeviceID_t      *acdDeviceID,
        DeviceID_t      *fwdToDeviceID);
```

Parameters

acsHandle

The handle for 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 identifiers in the **acsOpenStream( )**. The parameter is ignored by the ACS Library when the Stream is set for library-generated identifiers.

deviceID

A pointer to the device identifier of the device containing the ACD supervisor.

forwardACDGroup

If set to **true**, then the ACD group is forwarded to another destination. If set to **false**, then ACD group forwarding is canceled.

acdDeviceID

A pointer to the device identifier of the device that contains the ACD group to be forwarded.

fwdToDeviceID

A pointer to the device identifier of the device to which the ACD group is to be forwarded.

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 **ETPSetACDGroupForwardConfEvent** message to ensure that the service request has been acknowledged and processed by the server and switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
A bad or unknown *acsHandle* was provided by the application.

ACSERR_STREAM_FAILED
A previously active ACS Stream has been abnormally aborted.

Comments

This service is only valid for a logical device in the Idle state.

ETPSetACDGroupForwardConfEvent

ETPSetACDGroupForwardConfEvent provides a positive response from the server for a previous **etpSetACDGroupForward( )** request.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

```
typedef struct {
                    ACSHandle_t         acsHandle;
                    EventClass_t        eventClass;
                    EventType_t          eventType;
               } ACSEventHeader_t;
                    typedef struct {
                    ACSEventHeader_t       eventHeader;
                    union {
                              struct {
                                        InvokeID_t     invokeID;
                                        union {

                                                  ETPSetACDGroupForwardConfEvent_t  setACDGrpFwd;
                                                  } u;
                                                  } etpConfirmation;

                              } event;
                              } ETPEvent_t;
               typedef  struct  ETPSetACDGroupForwardConfEvent  _t {
                         Nulltype          null;
               } ETPSetACDGroupForwardConfEvent  _t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPCONFIRMATION**, which identifies this message as an ETP confirmation event.

eventType

Tag value ETP_SET_ACD_GROUP_FORWARD_CONF, which identifies this message as an ETPSetACDGroupForwardConfEvent.

invokeID

Specifies the function service request instance for the service that was processed at the server or switch. This identifier is provided to the application when a service request is made.

VIRTUAL DEVICE CALL CONTROL SERVICES

This section defines the Call Control Services supported for all virtual device types. These functions allow client applications to:

- Establish, control, and destroy calls at a virtual device
- Answer incoming calls at a virtual device

The following functions have additional functionality compared to how they work for the physical device type as defined in the *ApplicationLink Application Programmer's Guide.* They are described in more detail below.

cstaAnswerCall( )

cstaClearConnection( )

cstaDeflectCall( )/etpDeflectCallWithData()

cstaMakeCall( )

The following functions are only supported for virtual devices, and are described in more detail below:

etpClearCallInQueue()

etpJoinCalls( )

etpSplitCalls( )

Each function in this section has an associated confirmation event message as per standard TSAPI services. See “Control Services” and “Status Reporting Services” respectively, in this reference and in *Netware® Telephony Services™ Release 2 - Telephony Services Application Programming Interface (TSAPI)* for more information on events.

cstaAnswerCall( )

Virtual devices do not have the physical capabilities necessary to answer calls alerting at them. Therefore, the **cstaAnswerCall( )** service may only be invoked on calls alerting at virtual devices after the devices have media resources allocated to them. Refer to **etpAllocateResources( )** for details.

cstaClearConnection( )

The **cstaClearConnection( )** releases the specified virtual device from the designated call. The connection is left in the Null state. Additionally, the CSTA connection identifier provided in the service request is released. A successful **cstaClearConnection( )** service also deallocates the resources allocated to that call.

cstaDeflectCall( )/etpDeflectCallWithData

The **cstaDeflectCall( )** and **etpDeflectCallWithData( )** services release the specified virtual device from the designated call. A successful **cstaDeflectCall( )** or **etpDeflectCallWithData( )** service also deallocates the resources allocated to that call.

cstaMakeCall( )

The **cstaMakeCall( )** service originates a call from a device (the originator or calling device that must be on the switch) to another device (the destination or called device). When the calling device is a virtual device, the **cstaMakeCall( )** service is extended using the private data mechanism to accept a resource handle. When establishing an outbound call from a virtual device, private data is set to the resource handle returned in the **ETPAllocateResourcesConfEvent** message after resources are allocated. The resources must not be attached to an existing call.

For more information, refer to Appendix C, “ETP CSTA Private Data,” ApplicationLink Application Programmer’s Guide, and Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI).

Syntax

The following structure shows only the relevant additional portions of the unions for this message. For a complete description of the event structure, refer to Appendix C, “ETP CSTA Private Data,” Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

```
typedef struct ETP_PD_ResourceHandle_t  { char
                                vendor[32]
        unsigned short          length;
        ETPResourceHandle_t     resourceHandle;
```

```
} ETP_PD_ResourceHandle_t;

        typedef struct ETPResourceHandle_t  {
        ServerID_t              resourceServer;
        DeviceID_t              resourceDevice;
        DeviceID_t              ownerDevice;
        ResourceCharacteristic_t       resourceId;
} ETPResourceHandle_t;
```

Parameters

vendor

Stores the manufacturer object identifier. For **ETP_PD_ResourceHandle**, the sequence is:

{0x2B, 0x0C, 0x02, 0x89, 0x3D, 0x28, 0x03, 0x0}

length

Length of the resource handle.

resourceHandle

A handle to some previously allocated resources.

resourceId

Id of the resource handle.

Comments

Virtual devices do not have the physical capabilities necessary to establish outbound calls from them. To establish an outbound call from a virtual device, an application must first allocate resources for it via the **etpAllocateResources( )** service.

Note the following concerning the structure and parameters:

- The resource handle must immediately follow the *length* parameter.
- It is assumed that the byte alignment for the structure is 1. In other words, there is no gap between *vendor* parameter and *length* parameter as well as *length* parameter and *resourceHandle* parameter.

etpClearCallInQueue ( )

This **etpClearCallInQueue ( )** request allows a user to clear a call in a Queue when no media resources are attached to the call.

Virtual devices do not have the physical capabilities necessary to answer calls alerting at them until the necessary resource are allocated. At this time the call which is there in the queue can be cleared using **etpClearCallInQueue()** request.

Syntax

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

RetCode_t etpClearCallInQueue (
      ACSHandle_t                    hReq,
      InvokeID_t                     invokeID,
      ConnectionID_t*                pQueuedCall);
```

Parameters

hReq

The handle for 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 identifiers in the **acsOpenStream( )**. The parameter is ignored by the ACS Library when the Stream is set for library-generated identifiers.

pQueuedCall

A pointer to the connection identifier of one of the queued calls.

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 **ETPClearCallInQueueConfEvent** message to ensure that the service request has been acknowledged and processed by the OAS system.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
A bad or unknown *acsHandle* was provided by the application.

ACSERR_STREAM_FAILED
A previously active ACS Stream has been abnormally aborted.

ETPClearCallInQueueConfEvent

ETPClearCallInQueueConfEvent provides a positive response from the server for a previous **etpClearCallInQueue( )** request.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct
        {
        ACSEventHeader_t                eventHeader;
        union {
        struct {
        InvokeID_t                      invokeID;
                union {

ETPClearCallInQueueConfEvent_t              clearCallInQueue;
                            } u;
                } etpConfirmation;
                } event;
         } ETPEvent_t;

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

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPCONFIRMATION**, which identifies this message as an ETP confirmation event.

eventType

Tag value ETP_CIEAR_CALL_IN_QUEUE_CONF, which identifies this message as an ETPClearCallInQueueConfEvent.

invokeID

Specifies the function services request instance for the service that was processed at the server or switch. This identifier is provided to the application when a service request is made.

etpJoinCalls( )

The **etpJoinCalls( )** service connects an inbound call in the Connected state with an outbound call in the Connected or Alerting state at the same virtual device. In other words, the **etpJoinCalls( )** service connects the speech path between the two calls.

Syntax

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

RetCode _t etpJoinCalls ( ACSHandle_t
                          acsHandle,
        InvokeID_t      invokeID, ConnectionID_t
        *firstConnection, ConnectionID_t
        *secondConnection);
```

Parameters

acsHandle

The handle for 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 identifiers in the **acsOpenStream( )**. The parameter is ignored by the ACS Library when the Stream is set for library-generated identifiers.

firstConnection

A pointer to the connection identifier of one of the calls to be joined.

secondConnection

A pointer to the connection identifier of the call to be joined with the first call.

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 **ETPJoinCallsConfEvent** message to ensure that the service request has been acknowledged and processed by the OAS system.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
A bad or unknown *acsHandle* was provided by the application.

ACSERR_STREAM_FAILED
A previously active ACS Stream has been abnormally aborted.

Comments

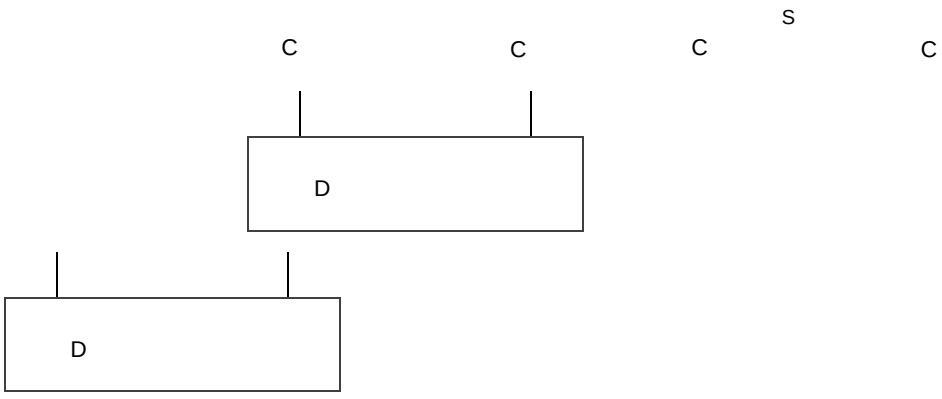
Any combination of inbound and outbound calls may be joined together with the following restrictions:

- Inbound calls must be in the Connected state.
- Outbound calls on a DSP Media Server must either be in the Connected or Alerting state.
- Outbound calls on an IP Media Server must be in the Connected state.

In addition, two calls at the same virtual device may be joined only if their call termination points coexist within the same physical media server (refer to **etpAllocateResources( )**).

Media activity cannot be initiated on a joined call.

A successful call to this function establishes a speech path between two calls at the same virtual device.



ETPJoinCallsConfEvent

Before

After

ETPJoinCallsConfEvent provides a positive response from the server for a previous **etpJoinCalls( )** request.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
            ACSEventHeader_t  eventHeader;
       union
       {


                struct
                {

                     InvokeID_t   invokeID;
                     union
                     {
                        ETPJoinCallsConfEvent_t   joinCalls;
                        } u;
                } etpConfirmation;
          } event;
} ETPEvent_t;
typedef struct ETPJoinCallsConfEvent_t  {
          Nulltype         null;
} ETPJoinCallsConfEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPCONFIRMATION**, which identifies this message as an ETP confirmation event.

eventType

Tag value **ETP_JOIN_CALLS_CONF**, which identifies this message as an **ETPJoinCallsConfEvent**.

invokeID

Specifies the function service request instance for the service that was processed at the server or switch. This identifier is provided to the application when a service request is made.

etpSplitCalls( )

etpSplitCalls( ) service disconnects the speech path between two calls previously joined at the

same virtual device using the **etpJoinCalls( )** service.

Syntax

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

RetCode_t etpSplitCalls ( ACSHandle_t
                          acsHandle,
        InvokeID_t      invokeID, ConnectionID_t
        *firstConnection, ConnectionID_t
        *secondConnection);
```

Parameters

acsHandle

The handle for 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 identifiers in the **acsOpenStream( )**. The parameter is ignored by the ACS Library when the Stream is set for library-generated identifiers.

firstConnection

A pointer to the connection identifier of one of the joined calls.

secondConnection

A pointer to the connection identifier of the call joined with the first call.

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 **ETPSplitCallsConfEvent** message to ensure that the service request has been acknowledged and processed by the OAS system.

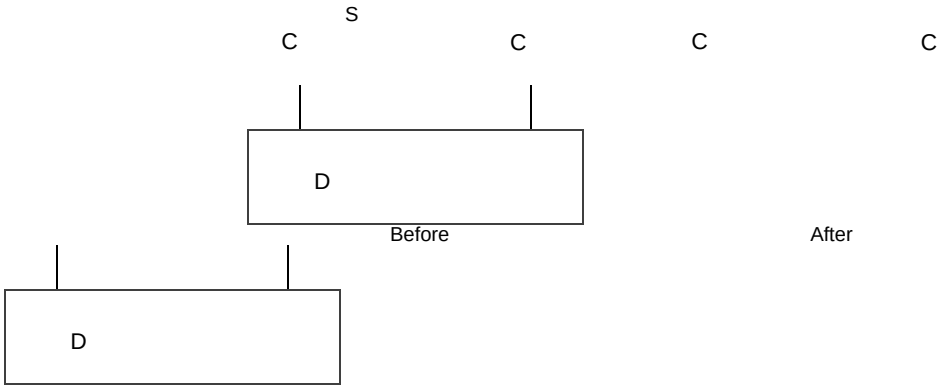
The following are possible negative error conditions for this function:

ACSERR_BADHDL
A bad or unknown *acsHandle* was provided by the application.

ACSERR_STREAM_FAILED
A previously active ACS Stream has been abnormally aborted.

Comments

A successful call to this function disconnects the speech path between two calls that were previously joined at the same virtual device using **etpJoinCalls( )**.



Before

ETPSplitCallsConfEvent

ETPSplitCallsConfEvent provides a positive response from the server for a previous **etpSplitCalls( )** request.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
           ACSEventHeader_t         eventHeader;
               union {
               struct
               {
               InvokeID_t      invokeID;
                         union
                         {
                       ETPSplitCallsConfEvent_t     splitCalls;
                         } u;
```

```
                    } etpConfirmation;
            } event;
    } ETPEvent_t;

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

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPCONFIRMATION**, which identifies this message as an ETP confirmation event.

eventType

Tag value **ETP_SPLIT_CALLS_CONF**, which identifies this message as an **ETPSplitCallsConfEvent**.

invokeID

Specifies the function service request instance for the service that was processed at the server or switch. This identifier is provided to the application when a service request is made.

INTER-APPLICATION COMMUNICATION DEVICE CONTROL SERVICES

The Inter-application Communication Device Control Services allow applications to communicate with each other. The applications use devices called Inter-application Communication Devices (ICDs) to send and receive messages asynchronously. Applications send messages to an ICD. To receive messages, an application must monitor an ICD and receive the **ETPReceivedMessageEvent** unsolicited event.

The device identifier of an ICD must begin with the tilde (~) character. An ICD is created by an application starting a monitor using **cstaMonitorDevice( )**. If the ICD does not exist, it is created at this time. When the application sends a message and the device exists, the ICD accepts the message. If the device does not exist, a negative confirmation event is returned. Multiple applications can receive messages from any ICD.

cstaMonitorDevice( )

The **cstaMonitorDevice( )** service initiates unsolicited event reporting from a device. An application can receive messages from an ICD by monitoring it via this service. In the **cstaMonitorDevice( )** service API call, the **ETP_PD_MonitorIcdFlag** private data can be passed to the OAS client library. Presence of private data in the **cstaMonitorDevice( )** signals OAS to start a monitor on an ICD and, if the device does not exist, the OAS dynamically creates the device.

Absence of this parameter means that if the device does not exist, the device will not be created and the parameter will be considered an invalid device identifier.

For more information, refer to Appendix C, “ETP CSTA Private Data,” Application Application Link Programmer’s Guide, and Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI).

Syntax

The following structure shows only the relevant additional portions of the unions for this message. For a complete description of the event structure, refer to Appendix C, “ETP CSTA Private Data,” Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

```
typedef struct ETP_PD_MonitorIcdFlag_t  {
        char    vendor[32];
        unsigned short length;
} ETP_PD_MonitorIcdFlag_t;
```

Parameters

vendor

Stores the manufacturer object identifier. For the **ETP_PD_MonitorIcdFlag**, the sequence is:

{0x2B, 0x0C, 0x02, 0x89, 0x3D, 0x28, 0x01, 0x0}

length

Length of the data portion of the private data structure that follows the *length* parameter. Since there is no data portion, *length* must contain **0** (zero).

Comments

Note the following concerning the structure and parameters:

It is assumed that the byte alignment for the structure is 1. In other words, there is no gap between the *vendor* parameter and the *length* parameter.

There is no data parameter.

etpSendMessage( )

The **etpSendMessage( )** service sends a message to an Inter-application Communication Device (ICD).

Syntax

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


RetCode_t etpSendMessage (
        ACSHandle_t             acsHandle,
        InvokeID_t              invokeID,
        DeviceID_t              *deviceID,
        Message_t               *message);
```

Parameters

acsHandle

The handle for 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 identifiers in the **acsOpenStream( )**. The parameter is ignored by the ACS Library when the Stream is set for library-generated identifiers.

deviceID

A pointer to the device identifier of the ICD to which the message is sent. The ICD device identifier must begin with the tilde (~) character.

message

A pointer to the message being sent on the ICD. This consists of the length of the message (maximum of 512 bytes) and the message data.

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 **ETPSendMessageConfEvent** message to ensure that the service request has been acknowledged and processed by the server and switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
A bad or unknown *acsHandle* was provided by the application.

ACSERR_STREAM_FAILED
A previously active ACS Stream has been abnormally aborted.

ETPSendMessageConfEvent

ETPSendMessageConfEvent provides a positive response from the server for a previous **etpSendMessage( )** request.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

```
typedef struct {
                        ACSHandle_t     acsHandle; EventClass_t
                                        eventClass; EventType_t
                                        eventType;
} ACSEventHeader_t;
typedef struct {
        ACSEventHeader_t        eventHeader;
                union {
                struct {
                           InvokeID_t      invokeID;
                                  union {
                                 ETPSendMessageConfEvent_t      sendMessage;
                                          } u;
                                  } etpConfirmation;
                                  } event;
           } ETPEvent_t;
typedef struct ETPSendMessageConfEvent_t  {
        Nulltype     null;
} ETPSendMessageConfEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPCONFIRMATION**, which identifies this message as an ETP confirmation event.

eventType

Tag value **ETP_SEND_MESSAGE_CONF**, which identifies this message as an **ETPSendMessageConfEvent**.

invokeID

Specifies the function service request instance for the service that was processed at the server or switch. This identifier is provided to the application when a service request is made.

Comments

The confirmation event indicates that the ICD to which the message was sent exists; (i.e., that it is being monitored). It does not, however, indicate the message has been received by any monitoring application or not.

MEDIA CONTROL SERVICES

The Media Control Services allow applications to provide media functions on connections at virtual devices *only*. Essentially, the application manipulates media resources to provide the necessary media functions.

The preconditions under which it is valid to invoke any of the Media Control Services (except for **etpAllocateResources( )** and **etpDeallocateResources( )**) are as follows:

- The target connection must contain a virtual device identifier.
- The virtual device's local connection state must be Connected, i.e., local connection state of **CS_CONNECT**.
- All the resources necessary to perform the requested function must have been allocated to the target connection.
- There must be no pending media activity on the target connection.

All the Media Control Services (except for **etpAllocateResources( )** and **etpDeallocateResources( )**) merely initiate the execution of media functions. A returned confirmation event indicates the requested function has initiated. Unsolicited events are generated during the execution of media functions initiated by Media Control Services. An application must use the related unsolicited events to track the execution of the requested media function.

etpAllocateResources( )

The **etpAllocateResources( )** service allocates the specified media resources to an existing call or a future outbound call.

Syntax

```
#include <acs.h>
#include <csta.h>
#include <etp.h>
RetCode_t etpAllocateResources (
        ACSHandle_t                         acsHandle,
        InvokeID_t                          invokeID,
        ConnectionID_t                      *call,
        ConnectionID_t                      *futureJoinedCall,
        ETPResourceList_t                   *resourceList,
        ETPUserDefinedCharacteristics_t     *userCharacteristics);
```

Parameters

acsHandle

The handle for 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 identifiers in the **acsOpenStream( )**. The parameter is ignored by the ACS Library when the Stream is set for library-generated identifiers.

call

A pointer to the connection identifier of the call to which resources are to be allocated. In the case of allocating resources for a future call, the call identifier contained in this parameter must be set to **ETP_NULL_CALL_ID**.

futureJoinedCall

This parameter is reserved for future functionality. It should be set to **ETP_NULL_CALL_ID**.

resourceList

A pointer to a list of the required resources to be allocated. A call termination point alone may be allocated by setting *resourceList* with no resources (blank). The capabilities of each resource type are as follows:

RESOURCE TYPE IDENTIFIER	CAPABILITY
asr	Automatic speech recognition used to recognize spoken words
signalDetector	Detect dual-tone multi-frequency (DTMF) digits or dialed pulse (DP) digits
signalGenerator	Send DTMF digits
player	Play pre-recorded sound specified as files or variables
recorder	Records caller voice into a Sound Media Object
ttsPlayer	Play text from TEXT file

Note: Player, asr and tts are language dependent.

userCharacteristics

A pointer to a null terminated string, containing up to **ETP_USER_DEFINED_CHARACTERISTIC_LENGTH** characters, that specifies a list of user-defined characteristics used as input to the resource allocation algorithm. When resource allocation is requested, the resource allocation algorithm is invoked to determine from which CTI server the system should obtain the requested resources. The value assigned to *userCharacteristics* is dependent upon the system configuration data. *userCharacteristics* contains call characteristics (specific characteristics of a call) and/or call requirements (requirements placed by a call on the CTI server). The syntax of the *userCharacteristics* parameter is specified (using Backus-Naur

Form) as follows:

<user_characteristics>	::= <characteristic_expression> | <characteristic_expression>,<user_characteristics>
<characteristic_expression>	::= <call_characteristic> | <call_requirement>
<call_characteristic>	::= <identifier>
<call_requirement>	::= <identifier>=<MANDATORY | PREFERRED>
<identifier>	::= <letter> | <digit> | <other_character> |<value><letter> | <value><digit> | <value><other_character>
<value>	::= <letter> | <digit> | <other_character> |<value><letter> | <value><digit> | <value><other_character>
<letter>	::= a | b | c | d | e | f | g | h | i | j | k | l | m |n | o | p | q | r | s | t | u | v | w | x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T |U | V | W | X | Y | Z
<digit>	::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<other_character>	::= - | _

The example below contains two call characteristics (`VIP_CUSTOMER` and `ROUTER_APP`), a mandatory call requirement (`LARGE_SERVER`), and a preferred call requirement (`FAST_SERVER`).

```
"VIP_CUSTOMER,FAST_SERVER=PREFERRED,ROUTER_APP,LARGE_SERVER=MANDATORY"
```

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 **ETPAllocateResourcesConfEvent** message to ensure that the service request has been acknowledged and processed by the server.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
A bad or unknown *acsHandle* was provided by the application.

ACSERR_STREAM_FAILED
A previously active ACS Stream has been abnormally aborted.

Comments

Resources exist within *physical media servers*. When resources are successfully allocated to a virtual device, a *call termination point* within one of the physical media servers is always allocated. This applies to both existing calls and future calls. When resources are reallocated to an existing call, the existing call termination point may be replaced with a new one within a different physical media resource node.

In the case of allocating resources to an existing call, the resources are connected to the specified connection. If the connection already has resources allocated to it, then a resource reallocation occurs; (i.e., the existing resources are deallocated and the new ones allocated). If the service does not complete successfully, then the previously allocated resources are unchanged. The service may only be invoked on an existing connection when either the connection is alerting with an inbound call or connected in speech with an inbound or outbound call. Resource *reallocation* on a connection with an inbound alerting call is not permitted. If resource reallocation is invoked on a connection having a media function active on it, then a successful completion of the service results in the media function being terminated; otherwise the media function is unaffected. Termination of an active media function as a result of resource allocation will produce the appropriate media unsolicited events.

As described under **cstaAnswerCall( )** and **cstaMakeCall( )**, virtual devices have only limited physical call handling capabilities. If an application wishes to perform any of the above services on a virtual device, it must first allocate at least a call termination point to it. A call termination point alone may be allocated by setting an empty *resourceList* parameter.

In the case of allocating resources to a future outbound call, the resources are merely reserved so that they may be used later for establishing an outbound call from a virtual device. The resource handle provided in **ETPAllocateResourcesConfEvent** is used in the **cstaMakeCall( )** service to establish the outbound call using the reserved resources. Resources allocated to a future call must have a call established on them within a particular timeout period. If not, then the resources are automatically deallocated and **ETPResourceTimeoutEvent** is sent. Resources allocated to a future call may not be reallocated. To achieve this, an application must first deallocate the resources via **etpDeallocateResources( )** and then allocate new resources. When resources are allocated for a future call but a **cstaMakeCall( )** fails and a **CSTAUniversalFailureEventConfEvent** is received, the allocated resources are not released and another **cstaMakeCall( )** can be made with the same resources. However, when a **cstaMakeCall( )** fails and a **cstaFailedEvent** is received, the resources allocated for the call are released.

When a call clears at a virtual device with resources allocated to it, the resources are automatically deallocated.

For information about **cstaAnswerCall( )** and **cstaMakeCall( )**, refer to the *ApplicationLink Application Programmer's Guide.* The other services and confirmation events are described in this chapter.

ETPAllocateResourcesConfEvent

ETPAllocateResourcesConfEvent provides a positive response from the server for a previous **etpAllocateResources( )** request, indicating the requested resources have been successfully allocated.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
               ACSEventHeader_t       eventHeader;
               union {
                struct {
                         InvokeID_t     invokeID;
                         union {
                         ETPAllocateResourcesConfEvent_t  allocateResources;
                                 } u;
                          } etpConfirmation;
               } event;
} ETPEvent_t;

typedef struct ETPAllocateResourcesConfEvent_t {
               ETPResourceHandle_t    resources;
} ETPAllocateResourcesConfEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPCONFIRMATION**, which identifies this message as an ETP confirmation event.

eventType

Tag value ETP_ALLOCATE_RESOURCES_CONF, which identifies this message as an ETPAllocateResourcesConfEvent.

invokeID

Specifies the function service request instance for the service that was processed at the server or switch. This identifier is provided to the application when a service request is made.

resourceHandle

A handle to the allocated resources.

etpDeallocateResources( )

The **etpDeallocateResources( )** service deallocates previously allocated resources.

Syntax

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

RetCode_t etpDeallocateResources  (
        ACSHandle_t             acsHandle,
        InvokeID_t              invokeID,
        ETPResourceHandle_t     *resources);
```

Parameters

acsHandle

The handle for 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 identifiers in the **acsOpenStream( )**. The parameter is ignored by the ACS Library when the Stream is set for library-generated identifiers.

resources

A handle to the resources to be deallocated. It is the handle returned in **ETPAllocateResourcesConfEvent** after resources are allocated.

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 **ETPDeallocateResourcesConfEvent** message to ensure that the service request has been acknowledged and processed by the server and switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
A bad or unknown *acsHandle* was provided by the application.

ACSERR_STREAM_FAILED
A previously active ACS Stream has been abnormally aborted.

Comments

The resources to be deallocated may or may not be attached to an existing call. Deallocating resources from a call also deallocates the call termination point from the call.

ETPDeallocateResourcesConfEvent

ETPDeallocateResourcesConfEvent provides a positive response from the server for a previous **etpDeallocateResources( )** request, indicating that the resources have been successfully deallocated.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
         ACSEventHeader_t    eventHeader;
        union {
                  struct {
                  InvokeID_t        invokeID;
                      union {
                      ETPDeallocateResourcesConfEvent_t  deallocateResources;
                             } u;
                      } etpConfirmation;
        } event;
} ETPEvent_t;
typedef struct ETPDeallocateResourcesConfEvent_t {
             Nulltype       null;
} ETPDeallocateResourcesConfEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPCONFIRMATION**, which identifies this message as an ETP confirmation event.

eventType

Tag value ETP_DEALLOCATE_RESOURCES_CONF, which identifies this message as an ETPDeallocateResourcesConfEvent.

invokeID

Specifies the function service request instance for the service that was processed at the server or switch. This identifier is provided to the application when a service request is made.

etpCollectDigits( )

The **etpCollectDigits( )** service initiates the collection of DTMF or DP digits from an established call in the Connected state. Optionally, it also provides the functions of the **etpPlay( )** service.

Syntax

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

RetCode_t etpCollectDigits  (
        ACSHandle_t               acsHandle,
        InvokeID_t                invokeID,
        ConnectionID_t            *call,
        ETPDigitDetectionType_t   detectionType,
        Boolean                   flushInputBuffer,
        int                       initialTimeout,
        int                       interDigitTimeout,
        int                       maxNumberOfDigits,
        ETPTerminationDigit_t     terminationDigits;
        Boolean                   interruptPlay,
        int                       messageId,
        ETPPlayList_t             *playList);
```

Parameters

acsHandle

The handle for 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 identifiers in the **acsOpenStream( )**. The parameter is ignored by the ACS Library when the Stream is set for library-generated identifiers.

call

A pointer to the connection identifier of the call from which the digit collection is to be applied.

detectionType

Specifies the required type of digit detection. The options are:

- DTMF detection (**ETP_DDT_DTMF**)
- DP detection (**ETP_DDT_DP**)
- Simultaneous DTMF and DP detection (**ETP_DDT_DTMFandDP**)
- Unknown **(ETP_DDT_UNKNOWN**)

flushInputBuffer

If set to **true**, then any previously detected and buffered digits will be deleted before digit detection is initiated; otherwise, the buffered digits are retained and considered as valid data.

initialTimeout

Specifies the timeout (in milliseconds) within which the first digit must be detected. The timeout starts after the play function ends or, if no play function was requested, the timeout starts immediately. Valid values are **0** to **ETP_MAX_INIT_DIGIT_TIMEOUT**, and **ETP_NO_TIMEOUT**.

interDigitTimeout

Specifies the timeout (in milliseconds) between which subsequent digits must be detected. Valid values are **0** to **ETP_MAX_INTER_DIGIT_TIMEOUT**, and **ETP_NO_TIMEOUT**.

maxNumberOfDigits

Specifies the maximum number of digits to be collected after which digit detection is terminated. The valid range is **1** to **ETP_MAX_DIGITS**.

terminationDigits

Specifies a null terminated string of digits.When one of these digits is detected, the collection is terminated. The valid values for the digits are:

- DP: ‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’
- DTMF: ‘0’, ‘1’, ‘2’, ‘3’, ‘4’, ‘5’, ‘6’, ‘7’, ‘8’, ‘9’, ‘*’, ‘#’

For both DP (Dial Pulse) and DTMF, the null string value signifies that no termination digit is enabled.

interruptPlay

This parameter is only meaningful if a *playList* is provided. If *interruptPlay* is set to **true**, then digit detection is enabled immediately and any detected digit will interrupt the play function. If set to **false**, then digit detection is enabled upon completion of the play function.

messageId

Refer to **etpPlay( )** for detailed information.

playList

Refer to **etpPlay( )** for detailed information. A null value along with a zero value for the messageId indicates that digit collection is to occur with no sound being played.

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 **ETPCollectDigitsConfEvent** message to ensure that the service request has been acknowledged and processed by the server and switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
A bad or unknown *acsHandle* was provided by the application.

ACSERR_STREAM_FAILED
A previously active ACS Stream has been abnormally aborted.

Comments

Before this service can be completed successfully, the resources to be allocated must include the *resourceList* parameter value **signalDetector** as per **etpAllocateResources( )**.

If a valid messageId or a non-null *playList* parameter are specified, then, in addition to the above specified resource, the resources are required to be allocated as per **etpPlay( )**.

When resources containing a signal detector are allocated, the signal detector detects DTMF digits entered by the caller even before **etpCollectDigits( )** is invoked for the first time on those resources. DP digit detection is turned off until an **etpCollectDigits( )** function is requested with *detectionType* set to **ETP_DDT_DP** or **ETP_DDT_DTMFandDP**.

When **etpCollectDigits( )** is requested with a specific *detectionType,* the *detectionType* remains in effect until the next request.

When **etpCollectDigits( )** is requested with *flushInputBuffer* set to **false**, the digits already entered by the caller and stored by the system will be returned in the **ETPCollectDigitsEndedEvent** even if the digits do not match the new request's *detectionType*.

A universal failure event can be received in case the request failed. The cause for the request failure can be one of the following:

FAILURE RESPONSE CAUSE	DESCRIPTION
UNKNOWN_MEDIA_PORT	Invalid or unknown call resource.
WRONG_MEDIA_PORT_STATE	If digit collector resource (or player resource if required) is in the wrong state.
NO_RESOURCE_ALLOCATED	No digit collector or player (if a play list or message is specified) resource allocated
INVALID_MAX_NUMBER_DIGITS	Max digits specified is out-of-range.
INVALID_PLAY_LIST	Invalid play list specified.
INVALID_DIGIT_DETECTION_TYPE	Requested digit detection type invalid. Valid values are ETP_DDT_DTMF, ETP_DDT_DP or ETP_DDT_DTMFandDP

ETPCollectDigitsConfEvent

ETPCollectDigitsConfEvent provides a positive response from the server for a previous **etpCollectDigits( )** request.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, "OAS Data Types," and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
        ACSEventHeader_t        eventHeader;
        union {
                struct {
                          InvokeID_t      invokeID;
                             union {
                              ETPCollectDigitsConfEvent_t      collectDigits;
                                    } u;
                           } etpConfirmation;
                } event;
} ETPEvent_t;
typedef struct ETPCollectDigitsConfEvent_t  {
        Nulltype        null;
```

```
} ETPCollectDigitsConfEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPCONFIRMATION**, which identifies this message as an ETP confirmation event.

eventType

Tag value **ETP_COLLECT_DIGITS_CONF**, which identifies this message as an **ETPCollectDigitsConfEvent**.

invokeID

Specifies the function service request instance for the service that was processed at the server or switch. This identifier is provided to the application when a service request is made.

etpDeleteMediaObject( )

The **etpDeleteMediaObject ( )** service deletes a Media Object.

Syntax

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

RetCode_t etpDeleteMediaObject  ( ACSHandle_t
                                  acsHandle,
        InvokeID_t                invokeID,
        DeviceID_t                *deviceID,
        ETPFileSpec_t   *fileSpec);
```

Parameters

acsHandle

The handle for 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 identifiers in the **acsOpenStream( )**. The parameter is ignored by the ACS Library when the Stream is set for library-generated identifiers.

deviceID

Indicates from which media server is the Media Object to be deleted.

fileSpec

Full path and name of the Media Object to be deleted.

Return Values

This function returns the following values depending on whether the application is using library-generated 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 **ETPDeleteMediaObjectConfEvent** message to ensure that the service request has been acknowledged and processed by the server and switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
A bad or unknown *acsHandle* was provided by the application.

ACSERR_STREAM_FAILED
A previously active ACS Stream has been abnormally aborted.

ETPDeleteMediaObjectConfEvent

ETPDeleteMediaObjectConfEvent provides a positive response from the server for a previous **etpDeleteMediaObject ( )** request.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

```
                            struct {
                            InvokeID_t        invokeID;
                                              union {
                            ETPDeleteMediaObjectConfEvent_t  deleteMediaObject;
                                           } u;
                                    } etpConfirmation;
                            } event
    } ETPEvent_t;
    typedef struct ETPDeleteMediaObjectConfEvent_t {
            Nulltype   null;
    } ETPDeleteMediaObjectConfEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPCONFIRMATION**, which identifies this message as an ETP confirmation event.

eventType

Tag value ETP_DELETE_MEDIA_OBJECT_CONF, which identifies this message as an ETPDeleteMediaObjectConfEvent.

invokeID

Specifies the function service request instance for the service that was processed at the server or switch. This identifier is provided to the application when a service request is made.

etpPlay( )

The **etpPlay( )** service initiates the playing of a list of Media Objects on an established call in the Connected state.

The play function has two behaviors:

- Simple Play Function: in this mode, all play objects are passed dynamically in a Play List. To invoke this play mode, the Play Message ID passed in the Play request must be set to zero
- Play Message Function: this mode initiates the playing of a pre-configured *play message*, that consists of a sequence of static and dynamic media objects. To invoke this play mode, the Play Message ID passed in the Play request must be set to a valid Message ID, as configured in the OAS Configuration database.

Syntax

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

```
RetCode_t etpPlay (
        ACSHandle_t     acsHandle,
        InvokeID_t      invokeID,
        ConnectionID_t  *call,
        int             messageId,
        char            *playList);
```

Parameters

acsHandle

The handle for 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 identifiers in the **acsOpenStream( )**. The parameter is ignored by the ACS Library when the Stream is set for library-generated identifiers.

call

A pointer to the connection identifier of the call to which the sound objects are to be played.

messageId

Identifies the particular message (specified by a number between 1 and 65535) within the system configuration database to be played. The message contains static (specified value) and/or dynamic (specified parameter) media objects.

playList

A pointer to a null terminated string containing up to ETP_PLAY_LIST_LENGTH characters (including the null character) that specifies a list of the play objects to be played. The list may contain play objects of the following types:

- TTS objects
  - Text Media Objects: The application specifies the name and path of the text media object to play
  - Text String: A text string passed dynamically
- Sound objects

  Pre-recorded sound file: <sound_media_object>. The application specifies the name and path of the sound media object to play, the offset in milliseconds from the beginning of the sound media object from which to begin play, and the duration in milliseconds of play within the media object.

  A positive, negative, or unsigned number containing up to

  ETP_MAX_PLAY_DIGITS digits (not including the sign character):

<signed_number>. For example, a number between -999999999999999 and +999999999999999.

DateDMY (day-month-year): <date_dmy> DateMDY

(month-day-year): <date_mdy> DateDM (day-month):

<date_dm> DateMD (month-day): <date_md>

Time12 (hour-minutes, 0:00 to 12:59 AM/PM): <time_hm> Time24

(hour-minutes, 0:00 to 23:59): <time_hm>

A string of characters up to **ETP_MAX_PLAY_CHARACTER_STRING_SIZE** characters to be "spelled out" character-by-character: <play_character_string>. The character string consists of 0-9, a-z, and A-Z.

Time duration in hours, minutes, and seconds (00:00:00 to 99:59:59): <duration>.

In the Simple Play Function mode, i.e. when the messageId is set to zero, the *playList* (<play_object_list>) is specified using the following syntax (defined in Backus-Naur Form):

<play_object_list>	::= <play_sound_list> | <play_tts_list>
<play_sound_list>	::= <play_sound_object>; | <play_sound_object>;<play_sound_list>
<play_sound_object>	::= SoundMediaObject=<sound_media_object> | CharString=<play_character_string> | Number=<signed_number> | DateDMY=<date_dmy> | DateMDY=<date_mdy> | DateDM=<date_dm> | DateMD=<date_md> | Time12=<time_hm> | Time24=<time_hm> | Duration=<duration>
<play_tts_object>	::= TtsFile=<tts_ media_object > | TtsString=<tts_string>
<tts_media_object>	::= <media_object_spec>
<tts_string>	::= <tts_character> | <tts_character><tts_string>
<sound_media_object>	::= <media_object_spec>,<offset_milliseconds>, <duration_millisconds>
<offset_milliseconds>	::= <number>
<duration_milliseconds>	::= <number>
<media_object_spec>	::= <media_object> | <media_container><media_object>
<media_object>	::= <character_string>
<media_container>	::= <character_string>: | <character_string>:<media_container>
<signed_number>	::= <number> | +<number> | -<number>
<number>	::= <digit> | <digit><number>
<date_dmy>	::= <digit><digit>-<digit><digit>- <digit><digit><digit><digit> Valid values are those of valid dates in day-month-year format.
<date_mdy>	::= <digit><digit>/<digit><digit>/ <digit><digit><digit><digit>

	Valid values are those of valid dates in month-day-year format.
<date_dm>	::= <digit><digit>-<digit><digit>
	Valid values are those of valid dates in day-month format.
<date_md>	::= <digit><digit>/<digit><digit>
	Valid values are those of valid dates in month-day format.
<time_hm>	::= <digit><digit>:<digit><digit>
	Valid values are those of valid time, 00:00 to 23:59.
<duration>	::= <digit><digit>:<digit><digit>:<digit><digit>
	Valid values are those of valid hours, minutes and seconds, 00:00:00 to 99:59:59.
<character_string>	::= <character><character_string>
<character>	::= <letter> | <digit> | <other_character>
<play_character_string>	::= <play_character><play_character_string>
<tts_character>	::= ASCII characters, as supported by the TTS
	resources
<letter>	::= a | b | c | d | e | f | g | h | i | j | k | l | m |n | o | p | q | r | s | t | u | v | w | x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T |U | V | W | X | Y | Z
<play_character>	::= a | b | c | d | e | f | g | h | i | j | k | l | m |n | o | p | q | r | s | t | u | v | w | x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T |U | V | W | X | Y | Z |0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
	Some letters may not be valid for certain languages.
<digit>	::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<other_character>	::= - | _

The following is an example specification of a list of sound objects:

```
"SoundMediaObject=EnglishVoices:phone_number_is,0,0; CharString=8425006;"
```

Invoking **etpPlay( )** on the above list will first play the entire (since an offset of 0 and duration of 0 are specified) pre-recorded sound file "EnglishVoices:phone_number_is" followed by the character string "8425006."

In the Play Message Function mode, i.e. when the messageId is set to a valid, non-zero value, the *playList* (<play_object_list>) is specified using the following syntax (defined in Backus-Naur Form):

<play_object_parameter_list>	::= <play_sound_parameter_objects> |
	<play_tts_parameter_objects>
<play_sound_parameter_objects	::= <play_sound_parameter_object>; |
>	<play_sound_parameter_object>;
	<play_sound_parameter_ objects>
<play_tts_parameter_objects>	::= <play_tts_parameter_object>; |
	<play_tts_parameter_object>;
	<play_tts_parameter_ objects>
<play_sound_parameter_object>	::= <sound_media_object> |

	<play_character_string> | <signed_number> | <number> | <date_dmy> | <date_mdy> | <date_dm> | <date_md> | <time_hm> | <time_hm> | <duration>
<sound_media_object > <signed_number> <number> <date_dmy> <date_mdy> <date_dm> <date_md> <time_12> <time_24> <duration> <tts_media_object> <tts_string>	(Same as described in previous table)

The following is an example of a play message entry in the configuration database. Note the following concerning the "Media Object Value" column:

- For static objects, the value is the name or value of the sound object to be played.
- For dynamic objects, the value indicates the position of the media object in the *playList* (which matches the parameter number in the selected *messageId*).

Note: This sound media object (Account_number_is) is specified in the play message as a relative path (relative to the default media container \EnglishPrompts\, which in turn is relative to the root container path <installation folder>\OAS\root_container.

1. The character string passed as the first element of the *PlayList* (ABC456) as "A B C four five six."
2. The sound stored in the media object:
   <installation folder>\OAS\root_container\ EnglishPrompts\Your_balance_is,0,0 (*0,0* means play the entire file)
3. The sound stored in the media object:
   <installation folder>\OAS\root_container\ EnglishPrompts\Your_balance_is,0,0 (*0,0* means play the entire file)

Note: This sound media object (Your_balance_is) is specified in the play message as a relative path (relative to the default media container \EnglishPrompts\, which in turn is relative to the root container path <installation folder>\OAS\root_container).

1. The number passed as the second element of the *playList* (123) as "one hundred twenty three."
2. The sound stored in the media object
   <installation folder>\OAS\root_container\US_currency\Dollars_and,0,0 (*0,0* means play the entire file)

Note: This sound media object (\US_currency\Dollars_and) is specified in the play message as a full path (relative to the root container path <installation folder>\OAS\root_container).

1. The number passed in the third element of *playList* (78) as “seventy eight.”
2. The sound stored in the media object:
   <installation folder>\OAS\root_container\US_currency\Cents,0,0 (*0,0* means play the entire file)

Note: This sound media object (\US_currency\Cents_and) is specified in the play message as a full path (relative to the root container path <installation folder>\OAS\root_container).

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 **ETPPlayConfEvent** message to ensure that the service request has been acknowledged and processed by the server and switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
A bad or unknown *acsHandle* was provided by the application.

ACSERR_STREAM_FAILED
A previously active ACS Stream has been abnormally aborted.

Comments

If a *playList* contains any combination of sound objects, a **player** resource must be allocated. If it contains any TTS objects, a **ttsPlayer** must be allocated.

Messages are configured and contained within the system configuration database. Each message consists of a series of pre-recorded sound or text objects. The database specifies the order in which the objects are to be played. These objects are either specified directly in the database (static) or by the parameter passed by the application (dynamic).

ETPPlayConfEvent

ETPPlayConfEvent provides a positive response from the server for a previous **etpPlay( )** request.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
        ACSEventHeader_t        eventHeader;
        union {
                struct {
                          InvokeID_t      invokeID;
                          union {
                                    ETPPlayConfEvent_t play;
                                    } u;
                    } etpConfirmation;
            } event;
       } ETPEvent_t;
typedef struct ETPPlayConfEvent_t
                { Nulltype  null;
} ETPPlayConfEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPCONFIRMATION**, which identifies this message as an ETP confirmation event.

eventType

Tag value **ETP_PLAY_CONF**, which identifies this message as an **ETPPlayConfEvent**.

invokeID

Specifies the function service request instance for the service that was processed at the server or switch. This identifier is provided to the application when a service request is made.

etpRecognize( )

The **etpRecognize( )** service initiates Automatic Speech Recognition on an established call in the Connected state. Optionally, it also provides the functions of the **etpPlay( )** service.

Syntax

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

RetCode_t etpRecognize (
        ACSHandle_t             acsHandle,
        InvokeID_t              invokeID,
        ConnectionID_t          *call,
        ETPGrammar              *grammar,
        Int                     initialTimeout,
        Int                     finalTimeout,
        Int                     maxTimeout,
        int                     numberOfResults,
        short                   interuptPlay,
        short           flushInputBuffer,
    ETPCallLogging_t             logging,
    int                                 messageId,
     ETPPlayList_t               *playList);
```

Parameters

acsHandle

The handle for 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 identifiers in the **acsOpenStream( )**. The parameter is ignored by the ACS Library when the Stream is set for library-generated identifiers.

call

A pointer to the connection identifier of the call from which the digit collection is to be applied.

grammar

String containing the name and path of the Grammar to be used for the recognition operation

initialTimeout

Specifies the initial silence timeout (in milliseconds) within which an utterance must be detected. The timeout starts after the play function ends or, if no message is to be played, immediately. Valid values are **ETP_ASR_MIN_INIT_TIMEOUT** to **ETP_ASR_MAX_INIT_TIMEOUT**.

finalTimeout

Specifies the silence timeout (in milliseconds)after the final utterance to indicate completion of recognition. Valid values are **ETP_ASR_MIN_FINAL_TIMEOUT** to **ETP_ASR_MAX_FINAL_TIMEOUT**.

maxTimeout

Specifies the maximum time (including the play function time, if applicable), where the recognize function is active. Valid values are **ETP_ASR_MIN_MAXIMUM_TIMEOUT** to **ETP_ASR_MAX_MAXIMUM_TIMEOUT**.

numberOfResults

Number of top choices to be returned by the Engine. Valid values are from – 1 to ETP_MAX_SPEECH_RESULTS.

interruptPlay

It is only meaningful if a play function is invoked. If *interruptPlay* is set to **true**, then the play function will be interrupted if:

- an utterance is detected, in which case word recognition is enabled immediately

  OR

- A DTMF tone is detected, in which case the word recognition is ended with a cause ETP_MEC_INTERRUPTED_BY_DIGIT.

If *interruptPlay* is set to **false**, then word recognition is enabled upon completion of the play function.

flushInputBuffer

Clear the DTMF buffer when the function is started. If this flag is set to **false** and one or more DTFM digits were stored in the buffer, then recognize functions will end as soon as it is started with cause ETP_MEC_INTERRUPTED_BY_DIGIT.

logging

Specifies whether the caller utterance should be recorded. Valid values are: 0 = no recording, 1 = record utterances.

messageId

Refer to **etpPlay( )** for detailed information.

playList

Refer to **etpPlay( )** for detailed information. A null value along with a zero value for the messageId indicates that digit collection is to occur with no sound being played.

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 **ETPRecognizeConfEvent** message to ensure that the service request has been acknowledged and processed by the server and switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
A bad or unknown *acsHandle* was provided by the application.

ACSERR_STREAM_FAILED
A previously active ACS Stream has been abnormally aborted.

Comments

Before this service can be completed successfully, the resources to be allocated must include the *resourceList* parameter value **asr**, and optionally a **signalDetector** if DTMF are also to be detected as per **etpAllocateResources( )**.

If a valid messageId or a non-null *playList* parameter are specified, then, in addition to the above specified resource, the resources are required to be allocated as per **etpPlay( )**.

See **etpCollectDigits()** for details on digit collection behavior when a signal detector has been allocated. If no signal detector has been allocated, then any digits entered should be ignored.

ETPRecognizeConfEvent

ETPRecognizeConfEvent provides a positive response from the server for a previous **etpRecognize( )** request.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {

                ACSEventHeader_t        eventHeader;
                union {
                          struct {
                                InvokeID_t      invokeID;
```

```
                                union {
                                        ETPRecognizeConfEvent_t   recognize;
                                        } u;
                                               } etpConfirmation;
                        } event;
        } ETPEvent_t;
typedef struct ETPRecognizeConfEvent  _t {
                Nulltype        null;
        } ETPRecognizeConfEvent  _t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPCONFIRMATION**, which identifies this message as an ETP confirmation event.

eventType

Tag value **ETP_RECOGNIZE_CONF**, which identifies this message as an **ETPRecognize ConfEvent**.

invokeID

Specifies the function service request instance for the service that was processed at the server or switch. This identifier is provided to the application when a service request is made.

etpRecord( )

The **etpRecord( )** service initiates the recording of a sound object on an established call in the Connected state.

Syntax

```
#include <acs.h>
#include <csta.h>
#include <etp.h> RetCode_t
etpRecord (
ACSHandle_t                acsHandle,
InvokeID_t                 invokeID,
ConnectionID_t         *call,
long                       maximumDuration, long
                        minimumDuration,
long                        silenceThreshold,
Boolean              stopOnDigitDetection,
ETPFileSpec_t            *fileSpec,
int                         beepFlag,
ETPMediaObjectsEncodingTypes_t  mediaObjectCodingType, int
                            overwriteFlag);
```

Parameters

acsHandle

The handle for 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 identifiers in the **acsOpenStream( )**. The parameter is ignored by the ACS Library when the Stream is set for library-generated identifiers.

call

A pointer to the connection identifier of the call to which the sound objects are to be played.

maximumDuration

The maximum duration, in milliseconds, allowed to record.

minimumDuration

If the recording function is terminated before the specified minimum duration (in milliseconds), e.g. caller hangs up, or silence detected, then recording function is considered to have failed.

silenceThreshold

If silence is detected for at least that duration (specified in milliseconds, the recording function is terminated.

stopOnDigitDetection

When this parameter's value is set to true, the recording will be terminated when a DTMF digit is detected. A signal detector must be already allocated before this function is called. The digit buffer will be cleared when this is set to TRUE.

fileSpec

This is the name and path of the Media Object where the recording is to be stored. This is optionally specified by the application. If the name is not specified by the application, then a Media Object name will be assigned by the Media Server and, that Media Object will be created in the Media Container set by the etpSetDefaultContainerPath, or the System default Container Path, if the etpSetDefaultContainerPath was not requested for the current call.

beepFlag

If this paramater is set to true, the caller will hear a beep before the recording starts.

mediaObjectCodingType

Indicates the format in which the recorded voice should be encoded.

overwriteFlag

If this paramter is set to true, and the Media Object in which to store the recording already exists, then the recording will overwrite the old recording in that Media Object. Otherwise the function will fail.

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 **ETPRecordEvent** message to ensure that the service request has been acknowledged and processed by the server and switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
A bad or unknown *acsHandle* was provided by the application.

ACSERR_STREAM_FAILED
A previously active ACS Stream has been abnormally aborted.

Comments

A universal failure event can be received in case the request failed. The cause for the request failure can be one of the following:

RESPONSE CAUSE	DESCRIPTION
UNKNOWN_MEDIA_PORT	If no associated call with the request
NO_RESOURCE_ALLOCATED	No recorder or digit collector (if ‘stop on digit detection’ is set) resource allocated
WRONG_MEDIA_PORT_STATE	If the recorder resource or associated call resource is not in the correct state for the request
INVALID_MEDIA_OBJECT	Improper file spec.
INVALID_MAX_DURATION	Maximum duration setting is out-of-range or is less than the minimum duration setting.
INVALID_MIN_DURATION	Minimum duration setting is out-of-range.
INVALID_SILENCE_THRESHOLD	Silence threshold setting is out-of-range.

ETPRecordConfEvent

ETPRecordConfEvent provides a positive response from the server for a previous **etpRecord( )** request.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

```
typedef struct {
        ACSHandle_t             acsHandle;
        EventClass_t            eventClass;
        EventType_t             eventType;
} ACSEventHeader_t;
typedef struct {
        ACSEventHeader_t        eventHeader;
        union {
            struct {
                      InvokeID_t    invokeID;
                                union {
                                  ETPRecordConfEvent_t  record;
                                          } u;
                      } etpConfirmation;
            } event;
} ETPEvent_t;
typedef struct ETPRecordConfEvent_t  {
        DeviceID_t      mediaRepositoryId;
        ETPFileSpec_t   fileSpec;
} ETPRecordConfEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPCONFIRMATION**, which identifies this message as an ETP confirmation event.

eventType

Tag value **ETP_RECORD_CONF**, which identifies this message as an **ETPRecordConfEvent**.

invokeID

Specifies the function service request instance for the service that was processed at the server or switch. This identifier is provided to the application when a service request is made.

mediaRepositoryId

Indicates the ID of the Media Repository.

fileSpec

Full name/path of created Media Object.

etpSetDefaultContainerPath( )

The **etpSetDefaultContainerPath ( )** service sets default container path to be appended to the beginning of each SoundMediaObject that starts with a colon (:) in any Media Functions such as etpPlay, etpPlayMessage, etpCollectWords, etc. The new value applies until another etpSetDefaultContainerPath or etpAllocateResource is called.

Syntax

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

RetCode_t etpSetDefaultContainerPath  ( ACSHandle_t
                                        acsHandle,
              InvokeID_t                 invokeID,
              ETPResrouceHandle_t            *resources,
              ETPDefaultContainerPath_t    *path);
```

Parameters

acsHandle

The handle for 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 identifiers in the **acsOpenStream( )**. The parameter is ignored by the ACS Library when the Stream is set for library-generated identifiers.

resources

A handle to which the default container path is to be set. It is the handle returned in **ETPAllocateResourcesConfEvent** after resources are allocated.

path

A null terminated string containing the new media container path to be set.

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 **ETPSetDefaultContainerPathConfEvent** message to ensure that the service request has been acknowledged and processed by the server and switch.

The following are possible negative error conditions for this function:

ACSERR_BADHDL
A bad or unknown *acsHandle* was provided by the application.

ACSERR_STREAM_FAILED
A previously active ACS Stream has been abnormally aborted.

Comments

The new value applies until another **etpSetDefaultContainerPath( )** or **etpAllocateResource( )** is called. Upon successful **etpAllocateResource( )**, the default container path is set to “User:”.

ETPSetDefaultContainerPathConfEvent

ETPSetDefaultContainerPathConfEvent provides a positive response from the server for a previous **etpSetDefaultContainerPath( )** request, indicating that the default container path has been successfully set. The default container path will be prepended to the beginning of each SoundMediaobject that starts with a colon (:) in a Media Function such as etpPlay, etpPlayMessage, etpCollectwords, etc.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
        ACSEventHeader_t      eventHeader;
         union {
                struct {
                          InvokeID_t      invokeID;
                          union {
                          ETPSetDefaultContainerConfEvent_t allocateResources;
                                  } u;
                          } etpConfirmation;
                 } event;
} ETPEvent_t;
```

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

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPCONFIRMATION**, which identifies this message as an ETP confirmation event.

eventType

Tag value ETP_SET_DEFAULT_CONTAINER_CONF, which identifies this message as an ETPSetDefaultContainerPathConfEvent.

invokeID

Specifies the function service request instance for the service that was processed at the server or switch. This identifier is provided to the application when a service request is made.

STATUS REPORTING SERVICES

This chapter includes descriptions of all OAS-extended CSTA unsolicited events as well as OAS-specific unsolicited events coming from the OAS system. In addition to the events described here, all events supported by the ApplicationLink product are also supported. See the *ApplicationLink Application Programmer's Guide* for details.

This chapter includes:

- Common Call Event Reports
- Inter-application Communication Device Event Reports
- Media Service Event Reports

COMMON CALL EVENT REPORTS

This section covers the OAS-extended CSTA unsolicited events and OAS-specific unsolicited events that can occur as a result of call activity on a device. Unless otherwise stated, the events are relevant to all types of devices. The events provide the application with call status information that can be used by the application in a variety of ways.

The following unsolicited events have additional functionality compared to how they are defined in ApplicationLink Application Programmer's Guide and Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI). They are described in more detail below.

- CSTAConferencedEvent
- CSTAConnectionClearedEvent
- CSTADeliveredEvent
- CSTADivertedEvent
- CSTAEstablishedEvent
- CSTAQueuedEvent
- CSTATransferredEvent

CSTAConferencedEvent

The **CSTAConferencedEvent** report indicates that two separate calls have been conferenced (merged) into a single call. This event is extended using the private data mechanism to accept **ETP_PD_AssociatedDataInfo**. This is the data previously associated with the call via the **etpAssociateData( )** service. For more information, refer to Appendix C, "ETP CSTA Private Data," *ApplicationLink Application Programmer's Guide*, and *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*.

Syntax

The following structure shows only the relevant additional portions of the unions for this message. For a complete description of the event structure, refer to Appendix C, "ETP CSTA Private Data," Chapter 7, "OAS Data Types," and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

```
 typedef struct ETP_PD_AssociatedDataInfo_t  {
char                  vendor[32]; unsigned
short                 length;
ETPAssociatedData_t   associatedData;
 } ETP_PD_AssociatedDataInfo_t;
```

Parameters

vendor

Stores the manufacturer object identifier. Must contain the following sequence:

{0x2B, 0x0C, 0x02, 0x89, 0x3D, 0x28, 0x02, 0x00}

length

The length of the associated data that follows the *length* parameter.

associatedData

A buffer of up to **MAX_LENGTH_ASSOCIATED_DATA** bytes that contains the associated data.

Comments

The associated data must immediately follow the *length* parameter.

It is assumed that the byte alignment for the structure is 1. In other words, there is no gap between the *vendor* parameter and the *length* parameter as well as the *length* parameter and the *associatedData* parameter.

CSTAConnectionClearedEvent

The **CSTAConnectionClearedEvent** report indicates that a device associated with a call disconnected or is dropped from the call. This event is extended using the private data mechanism to accept **ETP_PD_AssociatedDataInfo**. This is the data previously associated with the call via the **etpAssociateData( )** service.

For more information, refer to Appendix C, “ETP CSTA Private Data,”ApplicationLink Application Programmer’s Guide, and Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI).

Syntax

The following structure shows only the relevant additional portions of the unions for this message. For a complete description of the event structure, refer to Appendix C, “ETP CSTA Private Data,” Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

```
typedef struct ETP_PD_AssociatedDataInfo_t  {
char                    vendor[32]; unsigned
short                   length;
ETPAssociatedData_t     associatedData;
} ETP_PD_AssociatedDataInfo_t;
```

Parameters

vendor

Stores the manufacturer object identifier. Must contain the following sequence:

{0x2B, 0x0C, 0x02, 0x89, 0x3D, 0x28, 0x02, 0x00}

length

The length of the associated data that follows the *length* parameter.

associatedData

A buffer of up to **MAX_LENGTH_ASSOCIATED_DATA** bytes that contains the associated data.

Comments

The associated data must immediately follow the *length* parameter.

It is assumed that the byte alignment for the structure is 1. In other words, there is no gap between the *vendor* parameter and the *length* parameter as well as the *length* parameter and the *associatedData* parameter.

CSTADeliveredEvent

The **CSTADeliveredEvent** report provides information about a call that is alerting (i.e., ringing) at a specific device. This event is extended using the private data mechanism to accept **ETP_PD_AssociatedDataInfo**. This is the data previously associated with the call via the **etpAssociateData( )** service.

For more information, refer to Appendix C, “ETP CSTA Private Data,”ApplicationLink Application Programmer’s Guide, and Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI).

Syntax

The following structure shows only the relevant additional portions of the unions for this message. For a complete description of the event structure, refer to Appendix C, “ETP CSTA Private Data,” Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

```
typedef struct ETP_PD_AssociatedDataInfo_t  {
char                    vendor[32]; unsigned
short                   length;
ETPAssociatedData_t     associatedData;
} ETP_PD_AssociatedDataInfo_t;
```

Parameters

vendor

Stores the manufacturer object identifier. Must contain the following sequence:

{0x2B, 0x0C, 0x02, 0x89, 0x3D, 0x28, 0x02, 0x00}

length

The length of the associated data that follows the *length* parameter.

associatedData

A buffer of up to **MAX_LENGTH_ASSOCIATED_DATA** bytes that contains the associated data.

Comments

The associated data must immediately follow the *length* parameter.

It is assumed that the byte alignment for the structure is 1. In other words, there is no gap between the *vendor* parameter and the *length* parameter as well as the *length* parameter and the *associatedData* parameter.

CSTADivertedEvent

The **CSTADivertedEvent** report identifies a call that has been deflected or diverted from a monitored device. This event is extended using the private data mechanism to accept **ETP_PD_AssociatedDataInfo**. This is the data previously associated with the call via the **etpAssociateData( )** service.

For more information, refer to Appendix C, “ETP CSTA Private Data,”ApplicationLink Application Programmer’s Guide, and Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI).

Syntax

The following structure shows only the relevant additional portions of the unions for this message. For a complete description of the event structure, refer to Appendix C, “ETP CSTA Private Data,” Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

```
typedef struct ETP_PD_AssociatedDataInfo_t char
                              vendor[32];
      unsigned short                length;

ETPAssociatedData_t   associatedData;
} ETP_PD_AssociatedDataInfo_t;
```

Parameters

vendor

Stores the manufacturer object identifier. Must contain the following sequence:

{0x2B, 0x0C, 0x02, 0x89, 0x3D, 0x28, 0x02, 0x00}

length

The length of the associated data that follows the *length* parameter.

associatedData

A buffer of up to **MAX_LENGTH_ASSOCIATED_DATA** bytes that contains the associated data.

Comments

The associated data must immediately follow the *length* parameter. It is assumed that the byte alignment for the structure is 1. In other words, there is no gap between the *vendor* parameter and the *length* parameter as well as the *length* parameter and the *associatedData* parameter.

CSTAEstablishedEvent

The **CSTAEstablishedEvent** report indicates that a device connects to a call. This event is extended using the private data mechanism to accept **ETP_PD_AssociatedDataInfo**. This is the data previously associated with the call via the **etpAssociateData( )** service.

For more information, refer to Appendix C, “ETP CSTA Private Data,” ApplicationLink Application Programmer’s Guide, and Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI).

Syntax

The following structure shows only the relevant additional portions of the unions for this message. For a complete description of the event structure, refer to Appendix C, “ETP CSTA Private Data,” Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

```
typedef struct ETP_PD_AssociatedDataInfo_t  { char
                                vendor[32]; unsigned
        short                   length;
        ETPAssociatedData_t     associatedData;
} ETP_PD_AssociatedDataInfo_t;
```

Parameters

vendor

Stores the manufacturer object identifier. Must contain the following sequence:

{0x2B, 0x0C, 0x02, 0x89, 0x3D, 0x28, 0x02, 0x00}

length

The length of the associated data that follows the *length* parameter.

associatedData

A buffer of up to **MAX_LENGTH_ASSOCIATED_DATA** bytes that contains the associated data.

Comments

The associated data must immediately follow the *length* parameter.

It is assumed that the byte alignment for the structure is 1. In other words, there is no gap

between the *vendor* parameter and the *length* parameter as well as the *length* parameter and the *associatedData* parameter.

CSTAQueuedEvent

The **CSTAQueuedEvent** report indicates that a call has been queued at an Automatic Call Distribution (ACD) group device. This event is extended using the private data mechanism to accept **ETP_PD_AssociatedDataInfo**. This is the data previously associated with the call via the **etpAssociateData( )** service.

For more information, refer to Appendix C, “ETP CSTA Private Data,” ApplicationLink Application Programmer’s Guide, and Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI).

Syntax

The following structure shows only the relevant additional portions of the unions for this message. For a complete description of the event structure, refer to Appendix C, “ETP CSTA Private Data,” Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

```
typedef struct ETP_PD_AssociatedDataInfo_t  {
char                    vendor[32]; unsigned
short                   length;
ETPAssociatedData_t     associatedData;
} ETP_PD_AssociatedDataInfo_t;
```

Parameters

vendor

Stores the manufacturer object identifier. Must contain the following sequence:

{0x2B, 0x0C, 0x02, 0x89, 0x3D, 0x28, 0x02, 0x00}

length

The length of the associated data that follows the *length* parameter.

associatedData

A buffer of up to **MAX_LENGTH_ASSOCIATED_DATA** bytes that contains the associated data.

Comments

The associated data must immediately follow the *length* parameter. It is assumed that the byte alignment for the structure is 1. In other words, there is no gap between the *vendor* parameter and the *length* parameter as well as the *length* parameter and the *associatedData* parameter.

CSTATransferredEvent

The **CSTATransferredEvent** report indicates that an existing call was transferred to another device and the device that transferred the call is no longer part of the call (the transferring device has been dropped from the call). This event is extended using the private data mechanism to accept **ETP_PD_AssociatedDataInfo**. This is the data previously associated with the call via the **etpAssociateData( )** service.

For more information, refer to Appendix C, "ETP CSTA Private Data," ApplicationLink Application Programmer's Guide, and Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI).

Syntax

The following structure shows only the relevant additional portions of the unions for this message. For a complete description of the event structure, refer to Appendix C, "ETP CSTA Private Data," Chapter 7, "OAS Data Types," and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

```
typedef struct ETP_PD_AssociatedDataInfo_t  {
char                    vendor[32]; unsigned
short                   length;
ETPAssociatedData_t     associatedData;
                        } ETP_PD_AssociatedDataInfo_t;
```

Parameters

vendor

Stores the manufacturer object identifier. Must contain the following sequence:

{0x2B, 0x0C, 0x02, 0x89, 0x3D, 0x28, 0x02, 0x00}

length

The length of the associated data that follows the *length* parameter.

associatedData

A buffer of up to **MAX_LENGTH_ASSOCIATED_DATA** bytes that contains the associated data.

Comments

The associated data must immediately follow the *length* parameter. It is assumed that the byte alignment for the structure is 1. In other words, there is no gap between the *vendor* parameter and the *length* parameter as well as the *length* parameter and the *associatedData* parameter.

INTER-APPLICATION COMMUNICATION DEVICE EVENT REPORTS

This section covers the unsolicited event reports that can occur as a result of monitoring Inter-application Communication Devices (ICDs).

ETPReceivedMessageEvent

A monitoring application receives the unsolicited **ETPReceivedMessageEvent** report when a message is sent on an ICD via the **etpSendMessage( )** service.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

```
typedef struct {
                ACSHandle_t             acsHandle;
                EventClass_t             eventClass;
                EventType_t             eventType;
        } ACSEventHeader_t    ;
typedef struct {
        ACSEventHeader_t        eventHeader;
        union {
                struct {
                        CSTAMonitorCrossRefID_t   monitorCrossRefID;
                                union {
                        ETPReceivedMessageEvent  _t    receivedMessage;
                            } u;
                    } ETPUnsolicitedEvent;
         } event;
 } ETPEvent_t;

typedef struct ETPReceivedMessageEvent_t  {
                Message_t               message;
} ETPReceivedMessageEvent  _t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value **ETP_RECEIVED_MESSAGE**, which identifies this message as an **ETPReceivedMessageEvent**.

monitorCrossRefID

The handle to the CSTA association with 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.

message

The message received from the monitored ICD. It consists of the length of the message and the message data itself.

Comments

Since applications on different platforms use different formats, the interpretation of the data is the responsibility of the applications.

MEDIA SERVICE EVENTS REPORTS

This section covers the unsolicited events that can occur as a result of media activity on a monitored virtual device. The events provide the application with media status information that can be used by the application in a variety of ways.

ETPCollectDigitsEndedEvent

The **ETPCollectDigitsEndedEvent** report indicates that digit collection has ended.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

```
typedef struct {
        ACSHandle_t             acsHandle;
        EventClass_t    eventClass;
                EventType_t             eventType;
} ACSEventHeader_t;
typedef struct {
        ACSEventHeader_t        eventHeader;
        union {
                  struct {
                       CSTAMonitorCrossRefID_t   monitorCrossRefID;
                         union {
                       ETPCollectDigitsEndedEvent_t  collectDigitsEnded;
                                } u;
                          } etpUnsolicited;
                 } event;
         } ETPEvent_t;
typedef  struct  ETPCollectDigitsEndedEvent_t  {
           ConnectionID_t       call;
           ETPEventCause_t              cause;
           char                 detectedTerminationDigit;
           ETPDigitsType_t              digitsType;
           ETPDigitsList_t              collectedDigits;
} ETPCollectDigitsEndedEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value **ETP_COLLECT_DIGITS_ENDED**, which identifies this message as an **ETPCollectDigitsEndedEvent**.

monitorCrossRefID

The handle to the CSTA association with 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.

call

The connection identifier of the call where digit collection ended.

cause

The reason for this event.

EVENT CAUSE	DESCRIPTION
ETP_MEC_INITIAL_TIMEOUT	No digits entered prior to initial timeout.
ETP_MEC_INTER_DIGIT_TIMEOUT	Interdigit timeout expired.
ETP_MEC_MAXIMUM_DIGITS	Maximum digits entered.
ETP_MEC_TERMINATION_DIGIT	Termination digit entered.
ETP_MEC_DISCONNECT	Call disconnected during collect digits.

detectedTerminationDigits

Indicates which one of the Termination digits specified in the Collect Digits request was detected if any. If no Termination Digits were detected, this parameter is set to **null**.

digitsType

Specifies the type of digits detected. The valid types are:

- DTMF digits (**ETP_DDT_DTMF**)
- DP digits (ETP_DDT_DP)

- DTMF and DP digits (ETP_DDT_DTMFandDP)
- Unknown digit type (ETP_DDT_UNKNOWN), which occurs under the following conditions:
  - When a call arrives at a media server and if the user entered DTMF or DP digits before the first **etpCollectDigits( )** function is called, these digits are stored in the media server buffer without indication of their type (i.e., DTMF or DP).
  - When a caller does not input digits before the *initialTimeout* occurs.

collectedDigits

A null terminated string containing the collected digits (if any), including the termination digit (if any). Each digit is represented by its corresponding character.

Comments

This event is generated after any OAS service that initiates the collection of digits from a call and the collection is successful.

ETPCollectDigitsFailedEvent

The **ETPCollectDigitsFailedEvent** report indicates that digit collection has failed.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

```
typedef struct {
                ACSHandle_t             acsHandle;
        EventClass_t    eventClass;
        EventType_t             eventType;
        } ACSEventHeader_t;
typedef struct {
                ACSEventHeader_t        eventHeader;
             union {
                          struct {
                          CSTAMonitorCrossRefID_t   monitorCrossRefID;
                       union {
                                    ETPCollectDigitsFailedEvent_t
collectDigitsFailed;
                             } u;
                             } etpUnsolicited;
                  } event;
        } ETPEvent_t;
        typedef struct ETPCollectDigitsFailedEvent_t  {
                        ConnectionID_t          call;
                        ETPEventCause_t         cause;
        } ETPCollectDigitsFailedEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value ETP_COLLECT_DIGITS_FAILED, which identifies this message as an ETPCollectDigitsFailedEvent.

monitorCrossRefID

The handle to the CSTA association with 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.

call

The connection identifier of the call on which digit collection failed.

cause

The reason for this event.

EVENT CAUSE	DESCRIPTION
ETP_MEC_PLAY_FAILED	Attempt to play fails.
ETP_MEC_INTERNAL_ERROR	Dialogic error attempting to start collecting digits.

Comments

This event is generated after any OAS service that initiates the collection of digits from a call and the collection has failed.

ETPCollectDigitsStartedEvent

The **ETPCollectDigitsStartedEvent** report indicates that digit collection has started.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
```

```
                ACSEventHeader_t        eventHeader;
                union {
                          struct {
                         CSTAMonitorCrossRefID_t   monitorCrossRefID;
                                   union {
                                           ETPCollectDigitsStartedEvent_t

        collectDigitsStarted;
                                   } u;
                              } etpUnsolicited;
                        } event;
                    } ETPEvent_t;

typedef struct ETPCollectDigitsStartedEvent_t  {
ConnectionID_t                 call;
} ETPCollectDigitsStartedEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value ETP_COLLECT_DIGITS_STARTED, which identifies this message as an ETPCollectDigitsStartedEvent.

monitorCrossRefID

The handle to the CSTA association with 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.

call

The connection identifier of the call where digit collection started.

Comments

This event is generated after any OAS service that initiates the collection of digits from a call.

ETPDeleteMediaObjectCompletedEvent

The **ETPDeleteMediaObjectCompletedEvent** report indicates that a Media Object has been deleted.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

```
typedef struct {
                ACSHandle_t     acsHandle;
        EventClass_t    eventClass; EventType_t
                        eventType;
        } ACSEventHeader_t;
          typedef struct {
                ACSEventHeader_t        eventHeader;
                union {
                         struct {
                         CSTAMonitorCrossRefID_t   monitorCrossRefID;
                         union {
                         ETPDeleteMediaObjectCompletedEvent_t mediaObjectDeleted;
                         } u;
                         } etpUnsolicited;
                         } event;
        } ETPEvent_t;
typedef  struct  ETPDeleteMediaObjectCompletedEvent_t
         {
        ETPFileSpec_t           mediaObjectName; } ETPDeleteMediaObjectCompletedEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value ETP_MEDIA_OBJECT_DELETED, which identifies this message as an ETPDeleteMediaObjectCompletedEvent.

monitorCrossRefID

The handle to the CSTA association with which this event is associated. This handle is typically chosen by OAS and should be used by the application as a reference to a specific established association.

mediaObjectName

The name of the Media Object which was deleted successfully.

Comments

This event is generated after any OAS service that initiates the deletion of a Media Object from a Media Repository and the deletion is successful.

The Media Repository in which the Media Object deletion was requested, must be monitored by the application in order for that application to receive this unsolicited event.

ETPDeleteMediaObjectFailedEvent

The **ETPDeleteMediaObjectFailedEvent** report indicates that deletion of a Media Object from a Media Repository has failed.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
                ACSEventHeader_t        eventHeader;
             union {
                        struct {
                        CSTAMonitorCrossRefID_t   monitorCrossRefID;
                        union {

                    ETPDeleteMediaObjectFailedEvent_t  mediaObjectDeleteFailed;
                                } u;
                           } etpUnsolicited;
                     } event;
        } ETPEvent_t;
        typedef struct ETPDeleteMediaObjectFailedEvent_t{
                ETPFileSpec_t                   mediaObjectName;
                ETPEventCause_t                         cause;
        } ETPDeleteMediaObjectFailedEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value ETP_MEDIA_OBJECT_DELETE_FAILED, which identifies this message as an ETPDeleteMediaObjectFailedEvent.

monitorCrossRefID

The handle to the CSTA association with which this event is associated. This handle is typically chosen by OAS and should be used by the application as a reference to a specific established association.

mediaObjectName

The name of the Media Object which deletion has failed.

cause

The reason for this event.

Comments

This event is generated after any OAS service that initiates the deletion of a Media Object from a Media Repository and the deletion has failed.

The Media Repository in which the Media Object deletion was requested, must be monitored by the application in order for that application to receive this unsolicited event.

ETPMediaObjectCreatedEvent

The **ETPMediaObjectCreatedEvent** report indicates that a Media Object has been created in a Media Repository.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

} ACSEventHeader_t;
typedef struct {
        ACSEventHeader_t        eventHeader;
                union {
                        struct {
                                CSTAMonitorCrossRefID_t   monitorCrossRefID;
union {     ETPMediaObjectCreatedEvent_tmediaObjectCreated;
                                } u;
                        } etpUnsolicited;
                } event;
        } ETPEvent_t;
        typedef struct ETPMediaObjectCreatedEvent_t{
                ETPFileSpec_t                   mediaObjectName;
                ETPEventCause_t                         cause;
        } ETPMediaObjectCreatedEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value **ETP_MEDIA_OBJECT_CREATED**, which identifies this message as an **ETPMediaObjectCreatedEvent**.

monitorCrossRefID

The handle to the CSTA association with which this event is associated. This handle is typically chosen by OAS and should be used by the application as a reference to a specific established association.

mediaObjectName

The name of the Media Object which was created.

cause

The reason for this event.

Comments

This event is generated after any OAS service that creates a Media Object, e.g. an etpRecord() request.

The Media Repository in which the Media Object is created, must be monitored by the application in order for that application to receive this unsolicited event.

ETPPlayEndedEvent

The **ETPPlayEndedEvent** report indicates that the playing of either media objects or a message on a connection has ended.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program*

Interface (TSAPI), the ACS Data Types and CSTA Data Types sections.

```
typedef struct {
ACSHandle_t            acsHandle; EventClass_t
                       eventClass; EventType_t
                              eventType;
} ACSEventHeader_t;
typedef struct {
        ACSEventHeader_t       eventHeader;
        union {
        struct {
                CSTAMonitorCrossRefID_t   monitorCrossRefID;
                  union {
                      ETPPlayEndedEvent_t   playEnded;
                      } u;
                      } etpUnsolicited;
        } event;
} ETPEvent_t;

typedef struct ETPPlayEndedInfo_t  {
        int            interruptedElementIndex;
        int            interruptedOffset;
  } ETPPlayEndedInfo_t;

typedef struct ETPPlayEndedEvent_t  {
        ConnectionID_t                call;
        ETPEventCause_t               cause;
        ETPPlayEndedInfo_t            info;
  } ETPPlayEndedEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value **ETP_PLAY_ENDED**, which identifies this message as an **ETPPlayEndedEvent**.

monitorCrossRefID

The handle to the CSTA association with 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.

call

The connection identifier of the call where the playing of sound ended.

cause

The reason for this event.

info

Provides supplementary information when sound playing ends or is interrupted. It can consist of two pieces of information.

- *interruptedElementIndex*. An index to the last element or the element that was interrupted (the first element being given an *index* of **0**). The element is either a sound object in a play list or an entry (static or dynamic) in a play message.
- *interruptedOffset*. The offset in milliseconds from the start of the play list where the play function ended or was terminated.

Comments

This event is generated after any OAS service that initiates the playing of sound on a call and the playing is successful.

ETPPlayFailedEvent

The **ETPPlayFailedEvent** report indicates that the playing of either sound objects or a message on a connection has failed.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

        typedef struct {
                ACSEventHeader_t        eventHeader;
                union {
                        struct {
                             CSTAMonitorCrossRefID_t   monitorCrossRefID;
                               union {
                                        ETPPlayFailedEvent_t      playFailed;
                                } u;
                        } etpUnsolicited;
                } event;
} ETPEvent_t;

typedef struct ETPPlayFailedEvent_t  {
                ConnectionID_t                  call;
        ETPEventCause_t                cause;
                int                             invalidElementIndex;
        } ETPPlayFailedEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value **ETP_PLAY_FAILED**, which identifies this message as an **ETPPlayFailedEvent**.

monitorCrossRefID

The handle to the CSTA association with 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.

call

The connection identifier of the call where the playing of sound has failed.

cause

The reason for this event.

invalidElementIndex

Provides the supplementary information when an invalid sound element is detected.

It provides an *index* to the invalid element (the first element being given an *index* of **0**). The element is either a media object in a play list or a variable in a message.

Comments

This event is generated after any OAS service that initiates the playing of sound on a call and the playing has failed.

ETPPlayStartedEvent

The **ETPPlayStartedEvent** report indicates that the playing of either sound objects or a message on a connection has begun.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program*

Interface (TSAPI), the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
        ACSEventHeader_t        eventHeader;
        union {
                struct {
                        CSTAMonitorCrossRefID_t   monitorCrossRefID;
                                  union {
                                      ETPPlayStartedEvent_t
        playStarted;
                                  } u;
                                  } etpUnsolicited;
                } event;
       } ETPEvent_t;
typedef struct ETPPlayStartedEvent_t  {
        ConnectionID_t                  call;
        ETPEventCause_t                 cause;
} ETPPlayStartedEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value **ETP_PLAY_STARTED**, which identifies this message as an **ETPPlayStartedEvent**.

monitorCrossRefID

The handle to the CSTA association with 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.

call

The connection identifier of the call that has sound being played on it.

cause

The reason for this event.

Comments

This event is generated after any OAS service that initiates the playing of sound on a call.

ETPRecognizeEndedEvent

The **ETPRecognizeEndedEvent** report indicates that **etpRecognize** request has ended successfully.

This event contains the Speech Recognition results which consist of the following inforation

- Speech results
- Interpretation of each result
- Natural Language information for each interpretation

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

```
        typedef struct {
        ACSHandle_t             acsHandle;
        EventClass_t    eventClass; EventType_t
                                eventType;
} ACSEventHeader_t;
typedef struct {
        ACSEventHeader_t        eventHeader;
        union {
                struct {
                        CSTAMonitorCrossRefID_t   monitorCrossRefID;
                          union {
                                ETPRecognizeEndedEvent_t   *recognizeEnded;
                                 } u;
                           } etpUnsolicited;
                       } event;
        } ETPEvent_t;

   typedef struct ETPRecognizeSpeechResult_t{
                  char   szSpeech[ETP_MAX_SPEECH_RESULT_LENGTH];
               int    nScore;
        short  nFirstInterpIndex;
                                // index into stInterps[] of first
                                // interpretation (a speech result can have
                                // multiple interpretations.
                                   // nFirstInterpIndex
                        // points to the first one in the chain.
                        // See next section for discussion of
                                //stInterps[]).
        short  nNumberOfInterps;
                                // the number of successive interpretations
                                   // (starting with
                       // nFirstInterpIndex)  in this speech result
} ETPRecognizeSpeechResult_t;
        typedef struct ETPInterpretation_t
                  // "Interpretation".    This is something like:

                           // "command" "call" 10
```

```
                        // "person" "john doe" 10
                      // "department" "marketing" 10
                        // "phone" "2675" 10
        {
        short  nFirstNlResult;
// index into stNlResults of first NL result in   the
                            // interpretation  (an interpretation  can have
multiple NL
                       // results. nFirstNlResult points to the first one in the
                       // chain.  See next section for discussion of
stNlResults

        short  nNumberOfNlResults;     // the number of successive
                                               //NL results (starting with
                                               // nFirstNlResult)  in this
interpretation
} ETPInterpretation_t;


typedef struct ETPRecognizeNlResult_t{
               char   szSlot[ETP_MAX_SLOT_LENGTH];
               char   szSlotValue[ETP_MAX_SLOT_VALUE_LENGTH];
        int    nScore;
} ETPRecognizeNlResult_t;
typedef struct ETPRecognizeEndedEvent_t  {
        ConnectionID_t  call;
        short   nNumberOfSpeechResults;
        ETPRecognizeSpeechResult_t     stSpeechResults[ETP_MAX_SPEECH_RESULTS];
        ETPInterpretation_t    stInterps[ETP_MAX_INTERPS]; ETPRecognizeNlResult_t
        stNlResults[ETP_MAX_NL_RESULTS];
               ETPEventCause_t        cause;
        Boolean        bExceededMaxSpeechResults;
        Boolean        bExceededMaxInterps; Boolean
                       bExceededMaxNlResults;
        Boolean         bExceededMaxSpeechResultLength;
        Boolean         bExceededMaxSlotLength; Boolean
                        bExceededMaxSlotValueLength;


        } ETPRecognizeEndedEvent_t;
```

Note: the ETPRecognizeEndedEvent_t member (see ETPRecognizeEndedEvent_t *recognizeEnded above) of the ETPEvent_t is declared as a pointer unlike other structures in this union. In the interest of communications efficiency, it was decided to make it a pointer since this is a very large structure

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value **ETP_RECOGNIZE_ENDED**, which identifies this message as an **ETPRecognizeEndedEvent**.

monitorCrossRefID

The handle to the CSTA association with 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.

call

The connection identifier of the call on which collection of words ended.

nNumberOfSpeechResults

Number of results returned by the ASR engine.

stSpeechResults[ETP_MAX_SPEECH_RESULTS]

List of transcriptions of each speech recognition result. Includes index into stInterps array.

stInterps[ETP_MAX_INTERPS]

Each speech recognition result can have multiple NL interpretations. Each entry in this array corresponds to a speech result has a starting index and range in the stNlResults array.

stNlResults[ETP_MAX_NL_RESULTS]

Each entry contains a slot, slot value and score. A subset of entries in this list is associated with each speech recognition result.

cause

ETP_MEC_COMPLETED

bExceededMaxSpeechResults

Set to true if the **stSpeechResults** array has been filled up and there were more speech results. Subsequent speech results and interpretations have been ignored.

bExceededMaxInterps

Set to true if the **stInterps** array is filled up. Further speech results can be added to the speech results list but will have no interpretations or NL results.

bExceededMaxNlResults

Set to true if the **stNlResults** array is filled up. Further speech results can be added to the speech results list but will have no NL results.

bExceededMaxSpeechResultLength

Set to true if a speech result string length exceeds the results field size (ETP_MAX_SPEECH_RESULT_LENGTH). The string has been truncated.

bExceededMaxSlotLength

Set to true if a slot string length exceeds the results field size (ETP_MAX_SLOT_LENGTH). The string has been truncated.

bExceededMaxSlotValueLength

Set to true if a slot value string length exceeds the results field size (ETP_MAX_SLOT_VALUE_LENGTH). The string has been truncated.

Comments

This event is generated after any OAS service has initiated the **etpRecognize** request and the speech recognition is successful.

ETPRecognizeFailedEvent

The **ETPRecognizeFailedEvent** report indicates that **etpRecognize** request has failed.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
       ACSEventHeader_t       eventHeader;
                        union {
                        struct {
                             CSTAMonitorCrossRefID_t   monitorCrossRefID;
                                union {
                         ETPRecognizeFailedEvent_t   recognizeFailed;
                                      } u;
                         } etpUnsolicited;
               } event;
} ETPEvent_t;
       typedef struct ETPRecognizeFailedEvent_t  {
       ConnectionID_t  call;
       ETPEventCause_t        cause;
       Boolean         bExceededMaxSpeechResults;
       Boolean         bExceededMaxInterps;
       Boolean         bExceededMaxNlResults;
       Boolean         bExceededMaxSpeechResultLength;
       Boolean         bExceededMaxSlotLength;
       Boolean         bExceededMaxSlotValueLength;
} ETPRecognizeFailedEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value **ETP_RECOGNIZE_FAILED**, which identifies this message as an **ETPRecognizeFailedEvent**.

monitorCrossRefID

The handle to the CSTA association with 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.

call

The connection identifier of the call where collection of words failed.

cause

The reason for this event.

bExceededMaxSpeechResults

Set to true if the **stSpeechResults** array has been filled up in building the related **ETPRecognizeEndedEvent** event and there were more speech results.

bExceededMaxInterps

Set to true if the **stInterps** array has been filled up in building the related **ETPRecognizeEndedEvent** event.

bExceededMaxNlResults

Set to true if the **stNlResults** array has been filled up in building the related **ETPRecognizeEndedEvent** event.

bExceededMaxSpeechResultLength

Set to true if a speech result string length exceeds the results field size (ETP_MAX_SPEECH_RESULT_LENGTH). The string has been truncated.

bExceededMaxSlotLength

Set to true if a slot string length exceeds the results field size (ETP_MAX_SLOT_LENGTH) in building the related **ETPRecognizeEndedEvent** event.

bExceededMaxSlotValueLength

Set to true if a slot value string length exceeds the results field size (ETP_MAX_SLOT_VALUE_LENGTH) in building the related **ETPRecognizeEndedEvent** event.

EVENT CAUSE	DESCRIPTION
ETP_MEC_STOPPED	Sent when the media server stops recognition: when the player fails at some point when get a clear call and the ASR state is processing or idle
ETP_MEC_INTERRUPTED_BY_DIGIT	When the recognize process is interrupted by DTMF
ETP_MEC_NO_SPEECH	When a stream ended message is received and we haven't detected the start of speech.
ETP_MEC_TOO_MUCH_SPEECH	When a stream ended message is received and we haven't detected the end of speech (a "SpeechEnded" event).
ETP_MEC_FAILED	Other recognition failure (Nuance Rec Client will log the cause).
ETP_MEC_RECOGNITION_REJECTED	When you get a recognized event but the result code is 'Reject'.
ETP_MEC_RECOGNITION_TOO_SLO W	When you get a recognized event but the result code is 'TooSlowTimeout'.
ETP_MEC_SPEECH_TOO_EARLY	When you get a recognized event but the result code is 'SpeechTooEarly'.
ETP_MEC_INVALID_GRAMMAR	Attempting to start recognition processing at the engine fails due to 'unknown grammar'.
ETP_MEC_NOT_RECOGNIZED	When the final timeout is reached, the ASR engine will try to match 'all' received utterances with the Grammar. If it cannot make a match, it will return Not Recognized.

Comments

This event is generated after any OAS service has initiated the **etpRecognize** request and the speech recognition has failed.

ETPRecognizeStartedEvent

The **ETPRecognizeStartedEvent** report indicates that **etpRecognize** request has started.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a

complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
        ACSEventHeader_t        eventHeader;
        union {
                struct {
                        CSTAMonitorCrossRefID_t         monitorCrossRefID;
                          union {
                                ETPRecognizeStartedEvent_t      recognizeStarted;
                                  } u;
                } etpUnsolicited;
        } event;
} ETPEvent_t;

typedef struct ETPRecognizeStartedEvent_t  {
                ConnectionID_t                  call;
                ETPEventCause_t                 cause;
} ETPRecognizeStartedEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value **ETP_RECOGNIZE_STARTED**, which identifies this message as an **ETPRecognizeStartedEvent**.

monitorCrossRefID

The handle to the CSTA association with 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.

call

The connection identifier of the call where the collection of words started.

cause

ETP_MEC_SUCCESSFUL.

Comments

This event is generated after any OAS service has initiated the **etpRecognize** request.

ETPRecordEndedEvent

The **ETPRecordEndedEvent** report indicates that the recording of a media object has ended, that the recorded duration exceeds the 'minimumDuration' value specified in the **etpRecord( )** request, and that a file containing the recorded sound was created

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, "OAS Data Types," and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
        ACSEventHeader_t        eventHeader;
        union {
                struct {
                CSTAMonitorCrossRefID_t   monitorCrossRefID;
                          union {
                                  ETPRecordEndedEvent_t   recordEnded;
                                  } u;
                          } etpUnsolicited;
                           } event;
} ETPEvent_t;
typedef struct ETPRecordEndedEvent_t
{
                ConnectionID_t          call;
        DeviceID_t                      mediaRepositoryId;
        ETPFileSpec_t           mediaObjectName; long
                                duration;
        ETPEventCause_t                 cause;
} ETPRecordEndedEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value **ETP_RECORD_ENDED**, which identifies this message as an **ETPRecordEndedEvent**.

monitorCrossRefID

The handle to the CSTA association with which this event is associated. This handle is typically chosen by OAS and should be used by the application as a reference to a specific established association.

call

The connection identifier of the call where the recording ended.

mediaRepositoryId

The cross reference ID of the Media Repository where the recording is stored

mediaObjectName

The name and path of the Media Object where the recording is stored

Duration

Time duration of play recording.

cause

The reason for this event.

EVENT CAUSE	DESCRIPTION
ETP_MEC_DETECTED_DIGIT	Digit detected during recording
ETP_MEC_DISCONNECT	Call cleared during recording
ETP_MEC_MAXIMUM_DURATION	Maximum recording duration reached
ETP_MEC_RESOURCE_ REALLOCATION	Resource reallocated during recording
ETP_MEC_SILENCE_DURATION	Maximum silence detected during recording

Comments

This event is generated after any OAS service that initiates the recording of sound on a call and the recording is successful.

ETPRecordFailedEvent

The **ETPRecordFailedEvent** report indicates that the recording of a sound object on a connection has failed.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program*

Interface (TSAPI), the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
        ACSEventHeader_t        eventHeader;
        union {
                struct {
                        CSTAMonitorCrossRefID_t   monitorCrossRefID;
                          union {
                                  ETPRecordFailedEvent_t    recordFailed;
                                  } u;
                             } etpUnsolicited;
                } event;
} ETPEvent_t;
typedef struct ETPRecordFailedEvent_t

{
        ConnectionID_t                  call;
        DeviceID_t                      mediaRepositoryId;
        ETPEventCause_t                 cause;
} ETPRecordFailedEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value **ETP_RECORD_FAILED**, which identifies this message as an **ETPRecordFailedEvent**.

monitorCrossRefID

The handle to the CSTA association with which this event is associated. This handle is typically chosen by OAS and should be used by the application as a reference to a specific established association.

call

The connection identifier of the call where the recording of sound has failed.

mediaRepositoryId

The cross reference ID of the Media Repository where the recording is stored.

cause

The reason for this event.

EVENT CAUSE	DESCRIPTION
ETP_MEC_CLOSE_MEDIA_ OBJECT_FAILED	Error closing recorded file
ETP_MEC_INTERNAL_ERROR	Attempt to open output file or recording started then failed – either due to Dialogic error
ETP_MEC_INVALID_MEDIA_ OBJECT_ENCODING_TYPE	Invalid media type specified
ETP_MEC_MEDIA_OBJECT_ ALREADY_EXISTS	File exists and no overwrite flag
ETP_MEC_INVALID_MEDIA_ OBJECT	Invalid path or filename.
ETP_MEC_MINIMUM_ DURATION	Recording ended before minimum duration, due to: Digit detected Call cleared Maximum silence detected
ETP_MEC_UNKNOWN	Error detected by Dialogic

Comments

This event is generated after any OAS service that initiates the recording of sound on a call and the recording has failed.

ETPRecordStartedEvent

The **ETP**Record**StartedEvent** report indicates that the Recording of either sound objects or a message on a connection has begun.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
        ACSEventHeader_t        eventHeader;
        union {
                struct {
                CSTAMonitorCrossRefID_t   monitorCrossRefID;
                union {
                        ETPRecordStartedEvent_t                 RecordStarted;
                          } u;
                  } etpUnsolicited;
        } event;
} ETPEvent_t;
```

```
typedef struct ETPRecordStartedEvent_t
{
        ConnectionID_t                  call;
        DeviceID_t                      mediaRepositoryId;
} ETPRecordStartedEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value **ETP_RECORD_STARTED**, which identifies this message as an **ETPRecordStartedEvent**.

monitorCrossRefID

The handle to the CSTA association with 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.

call

The connection identifier of the call that has sound being recorded on it.

mediaRepositoryId

The cross reference ID of the Media Repository where the recording is stored.

Comments

This event is generated after any OAS service that initiates the recording of sound on a call.

ETPResourceTimeoutEvent

The **ETPResourceTimeoutEvent** report indicates that previously allocated resources for a future outbound call have not been attached to a call within the timeout period.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the

Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI), the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
        ACSEventHeader_t        eventHeader;
        union {
        struct {
                CSTAMonitorCrossRefID_t   monitorCrossRefID;
                union {
                        ETPResourceTimeoutEvent_t       resourceTimeout;
                          } u;
                   } etpUnsolicited;
        } event;
} ETPEvent_t;

typedef struct ETPResourceTimeoutEvent_t  {
        ETPResourceHandle_t     resources;
} ETPResourceTimeoutEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value **ETP_RESOURCE_TIMEOUT**, which identifies this message as an **ETPResourceTimeoutEvent**.

monitorCrossRefID

The handle to the CSTA association with 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.

resources

A handle to the timed-out resources.

Comments

After this event is generated, the resources are automatically deallocated.

ETPResourcesAllocatedEvent

The **ETPResourcesAllocatedEvent** report indicates that an application has successfully allocated resources to an existing call via the **etpAllocateResources( )** service.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
        ACSEventHeader_t        eventHeader;
        union {
        struct {
        CSTAMonitorCrossRefID_t   monitorCrossRefID;
                union {
                ETPResourcesAllocatedEvent_t   resourcesAllocated;
                          } u;
                } etpUnsolicited;
        } event;
} ETPEvent_t;

typedef struct ETPResourcesAllocatedEvent_t  { ConnectionID_t call;
        ETPResourceList_t                               resourceList;
        ETPResourceHandle_t                             resources;
        ETPEventCause_t                                         cause;
} ETPResourcesAllocatedEvent;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value **ETP_RESOURCES_ALLOCATED**, which identifies this message as an **ETPResourcesAllocatedEvent**.

call

The connection to which the resources were allocated.

resourceList

The list of resources allocated to the call (i.e., the list of required resources specified in **etpAllocateResources( )** to be allocated).

resources

A handle to the timed-out resources.

cause

The reason for this event.

Comments

The event is sent in the following scenarios:

An application has successfully allocated resources to an existing call via the **etpAllocateResources( )** service.

An application has successfully reallocated resources to an existing call. Note that the **ETPResourcesDeallocatedEvent** event is *not* sent in this scenario.

ETPResourcesDeallocatedEvent

The **ETPResourcesDeallocatedEvent** report indicates that the resources an application has successfully allocated to an existing call via the **etpAllocateResources( )** service have been deallocated.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

```
typedef struct {
                ACSHandle_t             acsHandle;
                EventClass_t    eventClass; EventType_t
                                        eventType;
        } ACSEventHeader_t;
        typedef struct {
                ACSEventHeader_t        eventHeader;
                union {
                        struct {
                                CSTAMonitorCrossRefID_t   monitorCrossRefID;
                                union {
                                ETPResourcesDeallocatedEvent_t
resourcesDeallocated;
                                 } u;
                        } etpUnsolicited;
                } event;
        } ETPEvent_t;
```

```
typedef struct ETPResourcesDeallocatedEvent_t  {
ConnectionID_t      call;
ETPEventCause_t     cause;
} ETPResourcesDeallocatedEvent;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value **ETP_RESOURCES_ALLOCATED**, which identifies this message as an **ETPResourcesAllocatedEvent**.

call

The connection from which the resources were deallocated.

cause

The reason for this event.

Comments

The event is sent in the following scenarios:

- An application has successfully deallocated resources from an existing call via the **etpDeallocateResources( )** service.
- An extension places a Basic Virtual Device (BVD) call on hold while having resources allocated to it. The call is placed back in the CTI group and, hence, the resources are lost.
- Under some conditions when a call is unsuccessfully deflected away from a BVD while having resources allocated to it.
- Under some conditions when resource reallocation across media servers or CTI servers fails due to the deflection failing.
- A fault condition has occurred, resulting in the resources being spontaneously deallocated from the call.

The event is *not* sent when a call is cleared down or diverted.

ETPSendDTMFEndedEvent

The **ETPSendDTMFEndedEvent** report indicates that the transmission of DTMFsignals on a call has ended.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
        ACSEventHeader_t        eventHeader;
        union {
                  struct {
                  CSTAMonitorCrossRefID_t   monitorCrossRefID;
                         union {
                         ETPSendDTMFEndedEvent_t    sendDTMFEnded;
                                   } u;
                         } ETPUnsolicitedEvent;
                  } event;
} ETPEvent_t;

typedef struct ETPSendDTMFEndedEvent_t  {
        ConnectionID_t                  call;
        ETPEventCause_t                 cause;
} ETPSendDTMFEndedEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value **ETP_SEND_DTMF_ENDED**, which identifies this message as an **ETPSendDTMFEndedEvent**.

monitorCrossRefID

The handle to the CSTA association with 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.

call

The connection identifier of the call where the transmission of DTMF signals has ended.

cause

The reason for this event.

Comments

This event is generated after any OAS service that initiates the transmission of a series of DTMF signals on a call and the transmission has ended.

ETPSendDTMFFailedEvent

The **ETPSendDTMFFailedEvent** report indicates that the transmission of DTMF signals on a call has failed.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
                ACSEventHeader_t        eventHeader;
                union {
                struct {
                        CSTAMonitorCrossRefID_t   monitorCrossRefID;
                          union {
                                ETPSendDTMFFailedEvent_t
        sendDTMFFailed;
                                } u;
                          } ETPUnsolicitedEvent;
                 } event;
} ETPEvent_t;

typedef struct ETPSendDTMFFailedEvent_t  { ConnectionID_t
                                                call;
                ETPEventCause_t                 cause;
} ETPSendDTMFFailedEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value **ETP_SEND_DTMF_FAILED**, which identifies this message as an **ETPSendDTMFFailedEvent**.

monitorCrossRefID

The handle to the CSTA association with 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.

call

The connection identifier of the call where the transmission of DTMF signals has failed.

cause

The reason for this event.

Comments

This event is generated after any OAS service that initiates the transmission of a series of DTMF signals on a call and the transmission has failed.

ETPSendDTMFStartedEvent

The **ETPSendDTMFStartedEvent** report indicates that the transmission of a series of DTMF signals on a call has begun.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
                ACSEventHeader_t        eventHeader;
                union {
                struct {
                        CSTAMonitorCrossRefID_t   monitorCrossRefID;
                          union {
                                 ETPSendDTMFStartedEvent_t
        sendDTMFStarted;
```

```
                                } u;
                              } ETPUnsolicitedEvent;
                              } event;
} ETPEvent_t;

typedef struct ETPSendDTMFStartedEvent_t  {
               ConnectionID_t                call;
} ETPSendDTMFStartedEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value **ETP_SEND_DTMF_STARTED**, which identifies this message as an **ETPSendDTMFStartedEvent**.

monitorCrossRefID

The handle to the CSTA association with 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.

call

The connection identifier of the call where the transmission of DTMF signals has started.

Comments

This event is generated after any OAS service that initiates the transmission of a series of DTMF signals on a call.

ETPSpeechDetectionEndedEvent

The **ETPSpeechDetectionEndedEvent** report indicates that the speech engine stopped collecting the utterance.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program*

Interface (TSAPI), the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
                ACSEventHeader_t        eventHeader;
                union {
                struct {
                        CSTAMonitorCrossRefID_t         monitorCrossRefID;
                          union {
                                ETPSpeechDetectionEndedEvent_t
                                speechDetectionEnded;
                           } u;
                             } etpUnsolicited;
                } event;
} ETPEvent_t;

typedef struct ETPSpeechDetectionEndedEvent_t  {
          ConnectionID_t        call;
         ETPEventCause_t        cause;
        long                    seconds;
         unsigned short         millisecs;
            short                 timezone;
            short                 dstflag;
} ETPSpeechDetectionEndedEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value **ETP_SPEECH_ENDED**, which identifies this message as an **ETPSpeechDetectionEndedEvent**.

monitorCrossRefID

The handle to the CSTA association with 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.

call

The connection identifier of the call where Speech detection ended.

cause

ETP_MEC_SUCCESSFUL.

seconds

Time in seconds since midnight (00:00:00), January 1, 1970, coordinated universal time (UTC).

millisecs

Fraction of a second in milliseconds.

timezone

Difference in minutes, moving westward, between UTC and local time.

dstflag

Nonzero if daylight savings time is currently in effect for the local time zone.

Comments

This event is generated after any OAS service has initiated the **etpRecognize** request and the user has completed his utterance.

ETPSpeechDetectionStartedEvent

The **ETPSpeechDetectionStartedEvent** report indicates that the speech engine has detected that the user has started an utterance.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct {
        ACSEventHeader_t        eventHeader;
        union {
                struct {
                        CSTAMonitorCrossRefID_t         monitorCrossRefID;
                        union {
                        ETPSpeechDetectionStartedEvent_t
                                                speechDetectionStarted;
                                } u;
```

```
                        } etpUnsolicited;
                } event;


} ETPEvent_t;
typedef struct ETPSpeechDetectionStartedEvent_t {
        ConnectionID_t  call;
        ETPEventCause_t   cause;
        long              seconds;
        unsigned short    millisecs;
        short             timezone;
        short             dstflag;
} ETPSpeechDetectionStartedEvent_t;
```

Parameters

acsHandle

The handle for the opened ACS Stream.

eventClass

Tag value **ETPUNSOLICITED**, which identifies this message as an ETP unsolicited event.

eventType

Tag value **ETP_SPEECH_STARTED**, which identifies this message as an **ETPSpeechDetectionStartedEvent**.

monitorCrossRefID

The handle to the CSTA association with 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.

call

The connection identifier of the call where Speech Detection started.

cause

ETP_MEC_SUCCESSFUL.

seconds

Time in seconds since midnight (00:00:00), January 1, 1970, coordinated universal time (UTC).

millisecs

Fraction of a second in milliseconds.

timezone

Difference in minutes, moving westward, between UTC and local time.

dstflag

Nonzero if daylight savings time is currently in effect for the local time zone.

Comments

This event is generated after any OAS service has initiated the **etpRecognize** request and the user has started his utterance

OAS DATA TYPES

For a listing of the data types used by the functions and events defined for the Open Application Server API, view the header files:

- etp.h: Contains the OAS event types, event structures, and function prototypes.
- etpdefs.h:Contains the OAS data types.
- etpmediadefs.h:Contains the OAS data types for Media Services.
- csta.h: Contains the OAS csta event types, event structures, and function prototypes
- cstadefs.h:Contains all the OAS csta event data types

STRUCTURES

CallingDeviceID_t:

```
typedef ExtendedDeviceID_t  CallingDeviceID_t;
```

CalledDeviceID_t:

```
typedef   ExtendedDeviceID_t  CalledDeviceID_t;
```

RedirectionDeviceID_t :

```
typedef    ExtendedDeviceID_t  RedirectionDeviceID_t;
```

SubjectDeviceID_t :

```
typedef    ExtendedDeviceID_t   SubjectDeviceID_t;
```

ConnectionID_t

CONNECTION IDENTIFIERS

A connection is the object that uniquely binds a call and a device. It is formed by combining a call identifier with a device identifier. Connection identifiers are used extensively in CSTA to make service requests. As defined in CSTA, a connection identifier may contain just a call identifier or just a device identifier. However, for Application Link, both the device identifier and the call identifier must be supplied for service requests.

The following is the structure for ConnectionID:

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

Refer to *Chapter 3 CSTA Services* in Application Link Programmer's Guide for information on how to make a connection.

ExtendedDeviceID_t

```
typedef struct ExtendedDeviceID_t  {
    DeviceID_t       deviceID;
    DeviceIDType_t   deviceIDType;
    DeviceIDStatus_t  deviceIDStatus;
} ExtendedDeviceID_t;
```

DeviceType_t

```
typedef enum DeviceType_t  {
    DT_STATION = 0,
    DT_LINE = 1,
    DT_BUTTON = 2,
    DT_ACD = 3, DT_TRUNK
    = 4, DT_OPERATOR =
    5,
    DT_STATION_GROUP  = 16,
    DT_LINE_GROUP  = 17,
    DT_BUTTON_GROUP  = 18,
    DT_ACD_GROUP  = 19,
    DT_TRUNK_GROUP  = 20,
    DT_OPERATOR_GROUP  = 21,
    DT_TSAPIEx = 253,       // New device for MTAP
    DT_OTHER = 255
} DeviceType_t;
```

CallType_t

```
typedef enum CallType_t  {
    CT_AUDIO = 0,
    CT_VIDEO = 1,
    CT_AUDIO_VIDEO  = 2
} CallType_t;
```

APPENDIX A TSAPI SERVICES SUPPORTED

The following tables list the TSAPI APIs and whether they are supported by the Open Application Server. The tables are:

- TSAPI Control Services and Confirmation Events
- TSAPI Switching Function Services and Confirmation Events TSAPI Status Reporting Services and Confirmation Events TSAPI Snapshot Reporting Services and Confirmation Events
- TSAPI CSTA Computing Function Services and Confirmation Events
- TSAPI Escape and Maintenance Services and Confirmation Events

In addition, note the following concerning TSAPI services supported by OAS:

OAS supports static connection ID devices, not dynamic connection ID devices. Therefore, **ConnectionID_Device** must always be set to **STATIC_ID**.

TSAPI Control Services and Confirmation Events

PROGRAM CALL	SUPPORTED	NOT SUPPORTED
acsAbortStream( )	X	
acsCloseStream( )	X	
ACSCloseStreamConfEvent	X	
acsOpenStream( )	X	
ACSOpenStreamConfEvent	X	
acsQueryAuthInfo( )		X
ACSUniversalFailureConfEvent	X	
ACSUniversalFailureEvent	X	
cstaGetAPICaps( )	X	
CSTAGetAPICapsConfEvent	X	
cstaGetDeviceList( )		X
CSTAGetDeviceListConfEvent		X

TSAPI Switching Function Services and Confirmation Events

PROGRAM CALL	SUPPORTED	NOT SUPPORTED
cstaAlternateCall( )	X	
CSTAAlternateCallConfEvent	X	
cstaAnswerCall( )	X	
CSTAAnswerCallConfEvent	X	
cstaCallCompletion( )	X	
CSTACallCompletionConfEvent	X	
cstaClearCall( )		X
CSTAClearCallConfEvent		X
cstaClearConnection( )	X	
CSTAClearConnectionConfEvent	X	
cstaConferenceCall( )	X	
CSTAConferenceCallConfEvent	X	
cstaConsultationCall( )	X	
CSTAConsultationCallConfEvent	X	
cstaDeflectCall( )	X	
CSTADeflectCallConfEvent	X	
cstaGroupPickupCall( )	X	
CSTAGroupPickupCallConfEvent	X	

cstaHoldCall( )	X	
CSTAHoldCallConfEvent	X	
cstaMakeCall( )	X	
CSTAMakeCallConfEvent	X	
cstaMakePredictiveCall( )	X	
CSTAMakePredictiveCallConfEvent	X	
cstaPickupCall( )	X	
CSTAPickupCallConfEvent	X	
cstaReconnectCall( )	X	
CSTAReconnectCallConfEvent	X	
cstaRetrieveCall( )	X	
CSTARetrieveCallConfEvent	X	
cstaTransferCall( )	X	
CSTATransferCallConfEvent	X	
cstaSetAgentState( )	X	
CSTASetAgentStateConfEvent	X	
cstaSetDoNotDisturb( )	X	
CSTASetDoNotDisturbConfEvent	X	
cstaSetForwarding( )	X	
CSTASetForwardingConfEvent	X	
cstaSetMsgWaitingInd( )		X
CSTASetMsgWaitingIndConfEvent		X
cstaQueryAgentState( )	X	
CSTAQueryAgentStateConfEvent	X	
cstaQueryDeviceInfo( )	X	
CSTAQueryDeviceInfoConfEvent	X	
cstaQueryDoNotDisturb( )	X	
CSTAQueryDoNotDisturbConfEvent	X	
cstaQueryForwarding( )	X	
CSTAQueryForwardingConfEvent	X	
cstaQueryLastNumber( )		X
CSTAQueryLastNumberConfEvent		X
cstaQueryMsgWaitingInd( )		X
CSTAQueryMsgWaitingIndConfEvent		X
CSTAUniversalFailureConfEvent	X	

TSAPI Status Reporting Services and Confirmation Events

PROGRAM CALL	SUPPORTED	NOT SUPPORTED
cstaMonitorDevice( )	X	
cstaMonitorCall( )		X
cstaMonitorCallsViaDevice( )		X
CSTAMonitorConfEvent	X	
cstaMonitorStop( )	X	
CSTAMonitorStopConfEvent	X	
cstaChangeMonitorFilter( )	X	
CSTAChangeMonitorFilterConfEvent	X	
CSTAMonitorEndedEvent	X	
CSTACallClearedEvent		X
CSTAConferencedEvent	X	
CSTAConnectionClearedEvent	X	
CSTADeliveredEvent	X	
CSTADivertedEvent	X	
CSTAEstablishedEvent	X	
CSTAFailedEvent	X	
CSTAHeldEvent	X	
CSTANetworkReachedEvent	X	
CSTAOriginatedEvent	X	
CSTAQueuedEvent	X	
CSTARetrievedEvent	X	
CSTAServiceInitiatedEvent	X	
CSTATransferredEvent	X	
CSTACallInfoEvent		X
CSTADoNotDisturbEvent	X	
CSTAForwardingEvent	X	
CSTAMessageWaitingEvent		X
CSTALoggedOnEvent	X	
CSTALoggedOffEvent	X	
CSTANotReadyEvent	X	
CSTAReadyEvent	X	
CSTAWorkNotReadyEvent	X	
CSTAWorkReadyEvent		X

TSAPI Snapshot Reporting Services and Confirmation Events

PROGRAM CALL	SUPPORTED	NOT SUPPORTED
cstaSnapshotCallReq( )		X
CSTASnapshotCallConfEvent		X
cstaSnapshotDeviceReq( )	X	
CSTASnapshotDeviceConfEvent	X	

TSAPI CSTA Computing Function Services and Confirmation Events

PROGRAM CALL	SUPPORTED	NOT SUPPORTED
cstaRouteRegisterReq( )		X
CSTARouteRegisterReqConfEvent		X
cstaRouteRegisterCancel( )		X
CSTARouteRegisterCancelConfEvent		X
CSTARouteRegisterAbortEvent		X
CSTARouteRequestEvent		X
CSTAReRouteEvent		X
cstaRouteSelect( )		X
CSTARouteUsedEvent		X
CSTARouteEndEvent		X
cstaRouteEnd( )		X

TSAPI Escape and Maintenance Services and Confirmation Events

PROGRAM CALL	SUPPORTED	NOT SUPPORTED
cstaEscapeService( )		X*
CSTAEscapeServiceConfEvent		X*
CSTAPrivateEvent		X*
CSTAPrivateStatusEvent		X*
CSTABackInServiceEvent	X	
CSTAOutOfServiceEvent	X	
cstaSysStatReq( )	X	
CSTASysStatReqConfEvent	X	
cstaSysStatStart( )	X	
CSTASysStatStartConfEvent	X	
cstaSysStatStop( )	X	
CSTASysStatStopConfEvent	X	
cstaChangeSysStatFilter( )	X	
CSTAChangeSysStatFilterConfEvent	X	
CSTASysStatEvent	X	

CSTASysStatEndedEvent	X

*In OAS, no escape services are defined. However, additional services (e.g., Media Services) are implemented as API calls.

APPENDIX B UNIVERSAL FAILURE EVENTS

Confirmation events defined for each service are sent as a positive response from the server for a previous service request. When the requested function service fails, an application can receive a universal failure event instead of a confirmation event. There are two types of universal failure events:

- CSTAUniversalFailureConfEvent
- ETPUniversalFailureConfEvent

CSTAUniversalFailureConfEvent

For information about the **CSTAUniversalFailureConfEvent**, refer to Chapter 5, “Switching Function Services,” in *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*.

ETPUniversalFailureConfEvent

The **ETPUniversalFailureConfEvent** provides a generic negative response from the server for a previously requested OAS (ETP) service. It is sent in place of any confirmation event when the requested function fails.

Syntax

The following structure shows only the relevant portions of the unions for this message. For a complete description of the event structure, refer to Chapter 7, “OAS Data Types,” and the *Netware® Telephone Services™ Release 2 - Telephony Services Application Program Interface (TSAPI)*, the ACS Data Types and CSTA Data Types sections.

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

typedef struct
{
        ACSEventHeader_t        eventHeader; Union {
        Struct {
                InvokeID_t      invokeID;
                union {
                        ETPUniversalFailureConfEvent_t

                                etpUniversalFailure;
                                } u;
                } etpConfirmation;
        } event;
} ETPEvent_t;

typedef struct ETPUniversalFailureConfEvent_t  {
```

```
          ETPUniversalFailure_t  error;
} ETPUniversalFailureConfEvent_t;


typedef enum ETPUniversalFailure_t  {
  ETP_ERROR_START=2000, BAD_PARAMETER,
  CCS_OUT_OF_SERVICE,          //Not used
        EMPTY_PLAY_LIST, INTERNAL_ERROR,
        INVALID_FINAL_TIMEOUT,
        INVALID_INITIAL_TIMEOUT,
        INVALID_INTER_DIGIT_TIMEOUT,
        INVALID_MAX_NUMBER_DIGITS,
        INVALID_MAX_NUM_WORDS,        //Not used
        INVALID_MIN_NUM_WORDS,        //Not used
        INVALID_NO_DIGIT_TIMEOUT, INVALID_PLAY_LANGUAGE,
        INVALID_PLAY_LIST, INVALID_TERMINATION_DIGIT,
        INVALID_VOCABULARY_ID,        //Not used
        MEDIA_PORT_OUT_OF_SERVICE,    //Not used
        NO_RESOURCE_ALLOCATED, RESOURCE_NOT_AVAILABLE,
        UNKNOWN_ETP_FAILURE,
        UNKNOWN_MEDIA_PORT, WRONG_MEDIA_PORT_STATE,
        INVALID_DIGIT_DETECTION_TYPE,
        INVALID_MAX_DURATION,
        INVALID_MEDIA_OBJECT_ENCODING_TYPE,
  INVALID_MEDIA_OBJECT,
       INVALID_MIN_DURATION,
  INVALID_SILENCE_THRESHOLD,
  MEDIA_OBJECT_DOES_NOT_EXIST,
        REMOTE_PARTY_BUSY,            //Not used
        OPEN_FILE_FAILED,
       OPEN_MEDIA_OBJECT_FAILED,
        CLOSE_MEDIA_OBJECT_FAILED,    //Not used
        READ_MEDIA_OBJECT_FAILED,
        WRITE_MEDIA_OBJECT_FAILED,
  DELETE_MEDIA_OBJECT_FAILED,  //Not used
        READ_FILE_FAILED,                     //Not  used
        WRITE_FILE_FAILED,                    //Not  used
        CLOSE_FILE_FAILED,                    //Not  used
        INCORRECT_RESOURCES,
       RESPONSE_TIMEOUT, INVALID_CALL_STATE,
        LICENSE_NOT_AVAILABLE,
        DEFLECT_ASSOCIATE_DATA_FAILED,
        RESOURCE_SERVER_LOADED, EMPTY_DTMF_LIST,
        INVALID_NUMBER_OF_RESULTS
} ETPUniversalFailure_t;
```

ERRORS

Error values indicate that an error has been returned while performing OAS functions. Specific error values are:

BAD_PARAMETER

One or more of the supplied parameter values are invalid.

CCS_OUT_OF_SERVICE

Reserved for future use.

CLOSE_MEDIA_OBJECT_FAILED

Unable to close the media object that was previously opened.

DEFLECT_ASSOCIATE_DATA_FAILED

Reserved for future use.

EMPTY_DTMF_LIST

A service was requested to collect digits with an empty *dtmflist*.

EMPTY_PLAY_LIST

A service was requested to play media objects with an empty *playList*.

INCORRECT_RESOURCES

A resource allocation service has been requested, but the resources requested are invalid or cannot be combined (e.g., **player**).

INTERNAL_ERROR

A service did not execute due to an internal error condition.

INVALID_CALL_STATE

An attempt was made to clear an unknown call.

INVALID_DIGIT_DETECTION_TYPE

A collect digits service was requested with an invalid digit detection type specified.

INVALID_FINAL_TIMEOUT

A service was requested to collect words with an invalid final timout parameter.

INVALID_INITIAL_TIMEOUT

A service was requested to collect words with an invalid initial timout parameter.

INVALID_INTER_DIGIT_TIMEOUT

A service was requested to collect digits with an invalid inter-digit timout parameter.

INVALID_MAX_DURATION

A record service was requested with a maximum duration out-of-range.

INVALID_MAX_NUMBER_DIGITS

A service was requested to collect digits with an invalid maximum number of digits parameter.

INVALID_MAX_NUM_WORDS

A service was requested to collect words with an invalid maximum number of words parameter.

INVALID_MEDIA_OBJECT

A service was requested to record with an invalid media object name.

INVALID_MEDIA_OBJECT_ENCODING_TYPE

A service was requested to record with an invalid media object type.

INVALID_MIN_DURATION

A record service was requested with a minimum duration out-of-range.

INVALID_MIN_NUM_WORDS

A service was requested to collect words with an invalid minimum number of words parameter.

INVALID_NO_DIGIT_TIMEOUT

A service was requested to collect digits with an invalid no digit timeout parameter.

INVALID_NUMBER_OF_RESULTS

A recognize service was requested with an invalid number of results specified.

INVALID_PLAY_LANGUAGE Reserved for future use. INVALID_PLAY_LIST

A service was requested to play media objects with an invalid *playList*.

INVALID_SILENCE_THRESHOLD

A record service was requested with an invalid silence threshold value.

INVALID_TERMINATION_DIGIT

A service was requested to collect digits with an invalid termination digit parameter.

INVALID_VOCABULARY_ID

A service was requested to collect words with an invalid vocabulary parameter.

LICENSE_NOT_AVAILABLE

Allocate Resource for ASR/TTS resource is requested but the license for that resource is not available.

MEDIA_OBJECT_DOES_NOT_EXIST

A service was requested on a media object which doesn't exist.

MEDIA_PORT_OUT_OF_SERVICE Reserved for future use. NO_RESOURCE_ALLOCATED

A media service was requested, but the necessary resources have not previously been allocated.

OPEN_FILE_FAILED

A service was requested on a media object that cannot be read to copy data from that media object.

OPEN_MEDIA_OBJECT_FAILED

A service was requested on a media object that cannot be opened.

READ_MEDIA_OBJECT_FAILED

A service was requested on a media object to which data cannot be copied.

REMOTE_PARTY_BUSY

A make call service was requested, but the remote party is not available.

RESOURCE_NOT_AVAILABLE

A resource allocation service has been requested, but the required resources are not available.

RESOURCE_SERVER_LOADED

No free channel is available in any suitable Media Server for this allocate resource request.

RESPONSE_TIMEOUT

The time within which a response is expected has expired.

WRITE_MEDIA_OBJECT_FAILED

A service was requested on a mediaobject to which data cannot be written.

UNKNOWN

A service did not execute due to an unknown error condition.

UNKNOWN_MEDIA_PORT

A service request was made on an unknown media port identifier.

WRONG_MEDIA_PORT_STATE

A service was requested when the media port is not in a state to process it.

APPENDIX C ETP CSTA PRIVATE DATA

Using ECMA CSTA's private data mechanism, Mitel has extended the standard features of CSTA to allow OAS applications to invoke OAS-specific features.

This appendix discusses:

- ECMA CSTA private data mechanism
- Manufacturer object
- identifier TSAPI private
- data structure ETP
- private data

ECMA CSTA PRIVATE DATA MECHANISM

As per the ECMA CSTA protocol ECMA-180, the private data mechanism includes

CSTAPrivateData, which is defined in ASN.1 (Abstract Syntax Notation 1) as follows:

```
CSTAProviderDATA  ::= [APPLICATION  30] IMPLICIT  SEQUENCE
{
        manufacturer            OBJECT  IDENTIFIER, ANY
        DEFINED  BY             manufacturer
}
```

For each private data, a unique manufacturer object identifier has been defined according to ISO 6523 object identifier recommendation.

For information about the OBJECT IDENTIFIER, see “Manufacturer Object Identifier.” For information about ANY DEFINED BY, see “TSAPI Private Data Structure.”

MANUFACTURER OBJECT IDENTIFIER

Each private data is identified by a unique object identifier. OBJECT IDENTIFIER data type is defined by ASN.1 syntax notation as follows:

ISO(1) identified-organization(3) icd-ecma(12) member-company(2)

ericsson(1213) Open Application Server(40) private(xx) The value *xx* in private data identifies the type of private data.

TSAPI PRIVATE DATA STRUCTURE

Each manufacturer object identifier defines a specific structure and the length of the structure.

Syntax

On the API level, PrivateData is defined by TSAPI as follows:

```
typedef  struct  PrivateData_t  {
        char                            vendor[32];
        unsigned  short         length;
        char                            data[1];
}  PrivateData_t;
```

Parameters

vendor

32-byte parameter that stores the manufacturer object identifier. The vendor parameter

contains the following:

```
vendor[0] = 0x2B;      /* iso(1) and identified-organization  (3) in one node*/
vendor[1] = 0x0C;      /* icd-ecma(12)  */
vendor[2] = 0x02;      /* member-company(2)  */
vendor[3] = 0x89;      /* ericsson  (first of two parts)  */
vendor[4] = 0x3D;      /* ericsson  (second of two parts)  */
vendor[5] = 0x28;      /* Open Application  Server(40)  */
vendor[6] =            /* defined for each type of private data */
vendor[7] = 0x00;   /* NULL terminator  */
```

length

The length of the data portion (if there is any) of the private data structure that follows the *length* parameter. If no data portion follows the *length* parameter, then the length must be specified as `0` (zero).

data

The *data* parameter, if there is any.

Example

To store the manufacturer object identifier for the private data in the **cstaMakeCall( )** service, the *vendor* parameter contains the following:

```
vendor[0] = 0x2B;      /* iso(1) and identified-organization  (3) in one node*/
vendor[1] = 0x0C;      /* icd-ecma(12)  */
vendor[2] = 0x02;      /* member-company(2)  */
vendor[3] = 0x89;      /* ericsson  (first of two parts)  */
vendor[4] = 0x3D;      /* ericsson  (second of two parts)  */
vendor[5] = 0x28;      /* Open Application  Server(40)  */
vendor[6] = 0x03;      /* specifies  that this is MakeCallVendorID  private
data*/
vendor[7] = 0x00;   /* NULL terminator  */
```

Comments

Note the following:

- For each extended CSTA service, confirmation, or unsolicited event, the exact structure for private data is specified.
- The structure must have the same exact parameters and lengths specified; otherwise, the data is considered invalid.

ETP PRIVATE DATA

The following services can pass private data to the OAS client library. See Chapter 5, “Switching Function Services,” for details.

cstaDeflectCall( )

cstaMakeCall( )

cstaMonitorDevice( )

The following unsolicited events can pass private data to the OAS client library. See Chapter 6, “Status Reporting Services,” for details.

CSTAConferencedEvent

CSTAConnectionClearedEvent

CSTADeliveredEvent

CSTADivertedEvent

CSTAEstablishedEvent

CSTAQueuedEvent

CSTATransferredEvent