Dialogic
Making Innovation Thrive

Dialogic® NaturalAccess™ Universal Speech Access API Developer's Manual

October 2009 64-0515-01

www.dialogic.com

Copyright and legal notices

All contents of this document are furnished for informational use only and are subject to change without notice and do not represent a commitment on the part of Dialogic Corporation or its subsidiaries ("Dialogic"). Reasonable effort is made to ensure the accuracy of the information contained in the document. However, Dialogic does not warrant the accuracy of this information and cannot accept responsibility for errors, inaccuracies or omissions that may be contained in this document.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH DIALOGIC® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN A SIGNED AGREEMENT BETWEEN YOU AND DIALOGIC, DIALOGIC ASSUMES NO LIABILITY WHATSOEVER, AND DIALOGIC DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF DIALOGIC PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY INTELLECTUAL PROPERTY RIGHT OF A THIRD PARTY.

Dialogic products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Due to differing national regulations and approval requirements, certain Dialogic products may be suitable for use only in specific countries, and thus may not function properly in other countries. You are responsible for ensuring that your use of such products occurs only in the countries where such use is suitable. For information on specific products, contact Dialogic Corporation at the address indicated below or on the web at www.dialogic.com.

It is possible that the use or implementation of any one of the concepts, applications, or ideas described in this document, in marketing collateral produced by or on web pages maintained by Dialogic may infringe one or more patents or other intellectual property rights owned by third parties. Dialogic does not provide any intellectual property licenses with the sale of Dialogic products other than a license to use such product in accordance with intellectual property owned or validly licensed by Dialogic and no such licenses are provided except pursuant to a signed agreement with Dialogic. More detailed information about such intellectual property is available from Dialogic's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. Dialogic encourages all users of its products to procure all necessary intellectual property licenses required to implement any concepts or applications and does not condone or encourage any intellectual property infringement and disclaims any responsibility related thereto. These intellectual property licenses may differ from country to country and it is the responsibility of those who develop the concepts or applications to be aware of and comply with different national license requirements.

Any use case(s) shown and/or described herein represent one or more examples of the various ways, scenarios or environments in which Dialogic® products can be used. Such use case(s) are non-limiting and do not represent recommendations of Dialogic as to whether or how to use Dialogic products.

Dialogic, Dialogic Pro, Brooktrout, Diva, Cantata, SnowShore, Eicon, Eicon Networks, NMS Communications, NMS (stylized), Eiconcard, SIPcontrol, Diva ISDN, TruFax, Exnet, EXS, SwitchKit, N20, Making Innovation Thrive, Connecting to Growth, Video is the New Voice, Fusion, Vision, PacketMedia, NaturalAccess, NaturalCallControl, NaturalConference, NaturalFax and Shiva, among others as well as related logos, are either registered trademarks or trademarks of Dialogic Corporation or its subsidiaries. Dialogic's trademarks may be used publicly only with permission from Dialogic. Such permission may only be granted by Dialogic's legal department at 9800 Cavendish Blvd., 5th Floor, Montreal, Quebec, Canada H4M 2V9. Any authorized use of Dialogic's trademarks will be subject to full respect of the trademark guidelines published by Dialogic from time to time and any use of Dialogic's trademarks requires proper acknowledgement.

Windows is a registered trademark of Microsoft Corporation in the United States and/or other countries. The names of actual companies and product mentioned herein are the trademarks of their respective owners.

This document discusses one or more open source products, systems and/or releases. Dialogic is not responsible for your decision to use open source in connection with Dialogic products (including without limitation those referred to herein), nor is Dialogic responsible for any present or future effects such usage might have, including without limitation effects on your products, your business, or your intellectual property rights.

Revision history

Revision	Release date	Notes
9000-62367-10	May 2004	MVH; SAI service 1.0 Beta
9000-62367-11	July 2004	MCM; SAI service 1.0
9000-62367-12	April 2005	MVH; SAI service 1.1 Beta
9000-62367-13	August 2005	MVH; SAI service 1.1
64-0515-01	October 2009	LBG, NaturalAccess R9.0
Last modified: September 21, 2009		

Refer to www.dialogic.com for product updates and for information about NMS support policies, warranty information, and service offerings.

Table Of Contents

1 Introduction

The *Dialogic® NaturalAccess™ Universal Speech Access API Developer's Manual* describes how to use the NaturalAccess Universal Speech Access API (USAI) to control MRCP services when developing applications that use automatic speech recognition (ASR) and text-to-speech (TTS) resources.

This manual also provides a detailed reference of Universal Speech Access API functions. This document defines telephony terms where applicable, but assumes that you are familiar with telephony concepts and the C programming language.

2 Terminology

Note: The product to which this document pertains is part of the NMS Communications Platforms business that was sold by NMS Communications Corporation ("NMS") to Dialogic Corporation ("Dialogic") on December 8, 2008. Accordingly, certain terminology relating to the product has been changed. Below is a table indicating both terminology that was formerly associated with the product, as well as the new terminology by which the product is now known. This document is being published during a transition period; therefore, it may be that some of the former terminology will appear within the document, in which case the former terminology should be equated to the new terminology, and vice versa.

Former terminology	Dialogic terminology
CG 6060 Board	Dialogic® CG 6060 PCI Media Board
CG 6060C Board	Dialogic® CG 6060C CompactPCI Media Board
CG 6565 Board	Dialogic® CG 6565 PCI Media Board
CG 6565C Board	Dialogic® CG 6565C CompactPCI Media Board
CG 6565e Board	Dialogic® CG 6565E PCI Express Media Board
CX 2000 Board	Dialogic® CX 2000 PCI Station Interface Board
CX 2000C Board	Dialogic® CX 2000C CompactPCI Station Interface Board
AG 2000 Board	Dialogic® AG 2000 PCI Media Board
AG 2000C Board	Dialogic® AG 2000C CompactPCI Media Board
AG 2000-BRI Board	Dialogic® AG 2000-BRI Media Board
NMS OAM Service	Dialogic® NaturalAccess™ OAM API
NMS OAM System	Dialogic® NaturalAccess™ OAM System
NMS SNMP	Dialogic® NaturalAccess™ SNMP API
Natural Access	Dialogic® NaturalAccess™ Software
Natural Access Service	Dialogic® NaturalAccess™ Service
Fusion	Dialogic® NaturalAccess™ Fusion™ VoIP API
ADI Service	Dialogic® NaturalAccess™ Alliance Device Interface API
CDI Service	Dialogic® NaturalAccess™ CX Device Interface API
Digital Trunk Monitor Service	Dialogic® NaturalAccess™ Digital Trunk Monitoring API
MSPP Service	Dialogic® NaturalAccess™ Media Stream Protocol Processing API
Natural Call Control Service	Dialogic® NaturalAccess™ NaturalCallControl™ API
NMS GR303 and V5 Libraries	Dialogic® NaturalAccess™ GR303 and V5 Libraries

Former terminology	Dialogic terminology
Point-to-Point Switching Service	Dialogic® NaturalAccess™ Point-to-Point Switching API
Switching Service	Dialogic® NaturalAccess™ Switching Interface API
Voice Message Service	Dialogic® NaturalAccess™ Voice Control Element API
NMS CAS for Natural Call Control	Dialogic® NaturalAccess™ CAS API
NMS ISDN	Dialogic® NaturalAccess™ ISDN API
NMS ISDN for Natural Call Control	Dialogic® NaturalAccess™ ISDN API
NMS ISDN Messaging API	Dialogic® NaturalAccess™ ISDN Messaging API
NMS ISDN Supplementary Services	Dialogic® NaturalAccess™ ISDN API Supplementary Services
NMS ISDN Management API	Dialogic® NaturalAccess™ ISDN Management API
NaturalConference Service	Dialogic® NaturalAccess™ NaturalConference™ API
NaturalFax	Dialogic® NaturalAccess™ NaturalFax™ API
SAI Service	Dialogic® NaturalAccess™ Universal Speech Access API
NMS SIP for Natural Call Control	Dialogic® NaturalAccess™ SIP API
NMS RJ-45 interface	Dialogic® MD1 RJ-45 interface
NMS RJ-21 interface	Dialogic® MD1 RJ-21 interface
NMS Mini RJ-21 interface	Dialogic® MD1 Mini RJ-21 interface
NMS Mini RJ-21 to NMS RJ-21 cable	Dialogic® MD1 Mini RJ-21 to MD1 RJ-21 cable
NMS RJ-45 to two 75 ohm BNC splitter cable	Dialogic® MD1 RJ-45 to two 75 ohm BNC splitter cable
NMS signal entry panel	Dialogic® Signal Entry Panel

3 Using WebHelp

To view this manual with full functionality, use one of the following browsers:

- Internet Explorer 6.0 or later
- Mozilla 1.4 or later
- Netscape 6.2 or later

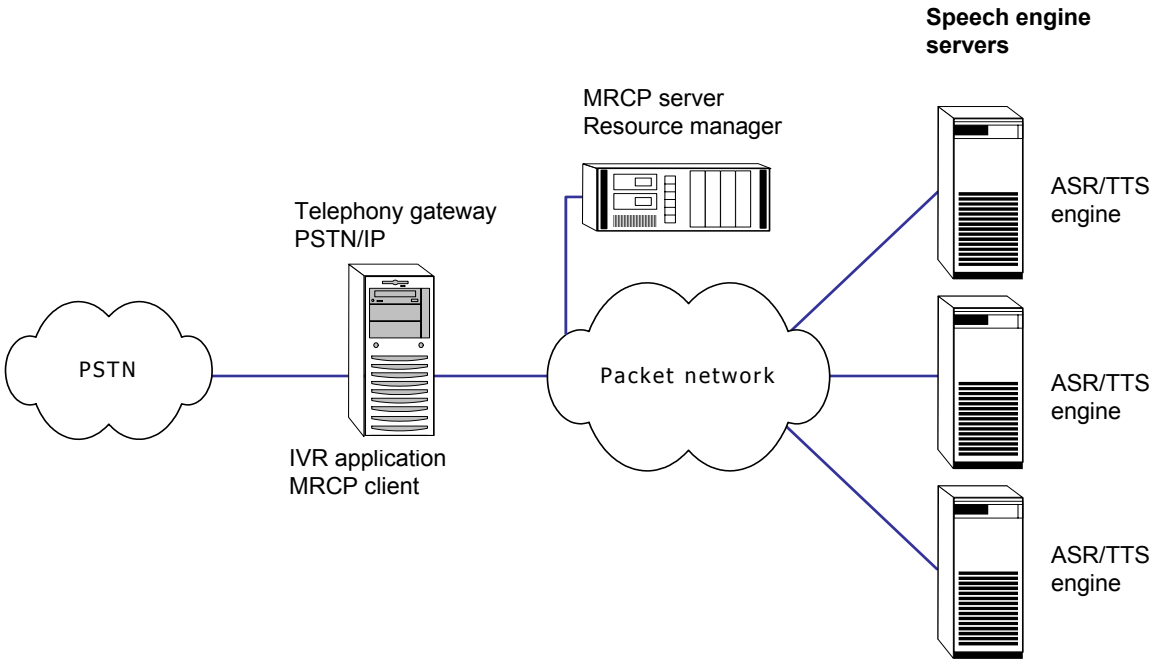
The following table describes the features in this manual:

4 Overview of the Universal Speech Access API

Media resource control protocol (MRCP)

Media resource control protocol (MRCP) is an application protocol for accessing automatic speech recognition (ASR) and text-to-speech (TTS) engines through IP networks. MRCP minimizes the resources needed to integrate various speech-based technologies for ASR and TTS platforms.

MRCP is designed for network-based solutions where ASR or TTS servers are configured to work together with VoiceXML interpreters, media gateways, and application servers. MRCP uses RTP (real time protocol) to transport audio information received from callers, as well as audio transferred to callers, from recordings or text-to-speech processes. The following illustration shows MRCP components:



The MRCP protocol does not specify how the control session is established with the server. It relies on real time streaming protocol (RTSP) to establish and maintain sessions. The session description protocol (SDP) is responsible for establishing media connections from clients to the network servers.

MRCP components

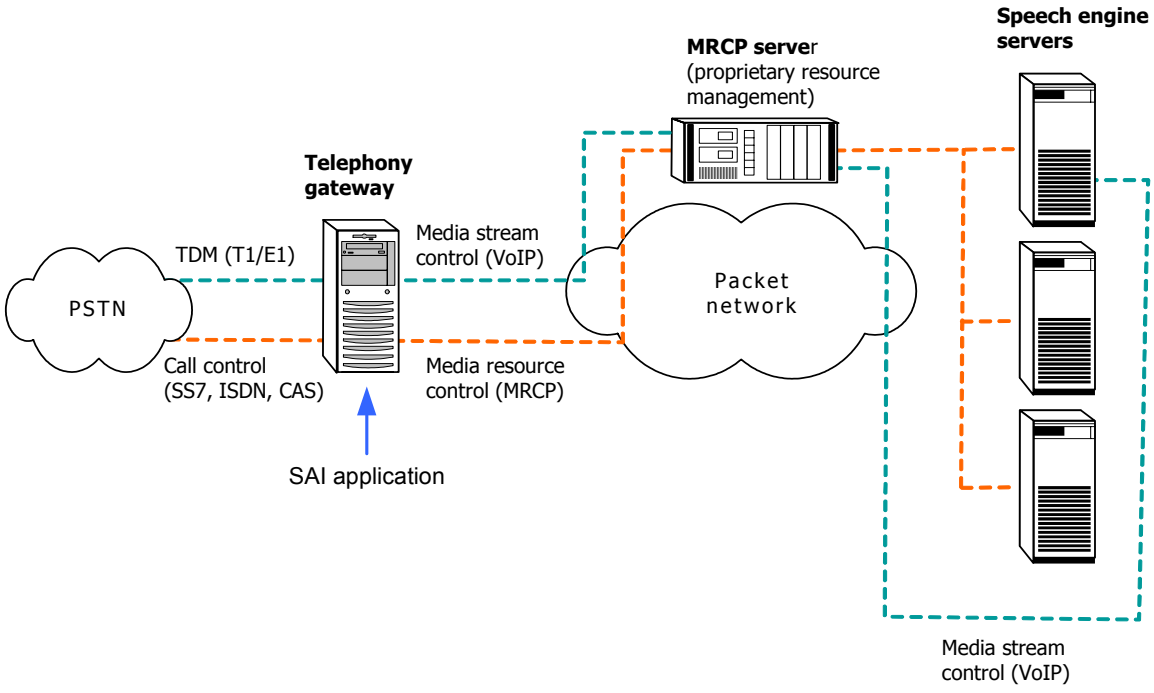
The MRCP architecture consists of the following components:

Component	Description
Clients	Provides media streams that can be generated or processed by ASR or TTS engines.
Servers	Provides resources or devices for processing or generating the streams. Examples include speech recognizers, speech synthesizers, speaker verification and speaker identification servers, and signal generators and detectors.

The MRCP protocol defines requests, responses, and events that control media processing resources and the state machine for each resource. The MRCP control architecture has the following characteristics:

- MRCP clients establish control sessions with servers for media processing by using a protocol such as RTSP. Clients also set up the appropriate RTP streams between the server and the client, allocating ports and setting up transport parameters as needed.
- Resources required for processing or generating media streams are referenced through URLs. Individual resources offered by MRCP servers can be referenced by resource URLs within a container URL.
- MRCP clients use MRCP messages to control media resources.

The following illustration shows MRCP media streaming and control:



Universal Speech Access API architecture

The Universal Speech Access API (USAI) enables you to implement MRCP services and resources using boards within the NaturalAccess environment. Applications use USAI to stream voice data from boards over RTP streams to recognizer engines and synthesizer engines on separate servers. Since the host processes no voice traffic, USAI improves the platform's bus and host processing capacity. In addition, voice activity detection (VAD) and pre-speech buffers on NaturalAccess boards reduce traffic to the ASR engines and decrease the number of required ASR ports.

NaturalAccess provides APIs for call control, system configuration, DTMF detection and tone generation, and other functions. The following table lists some of the NaturalAccess APIs that USAI applications use:

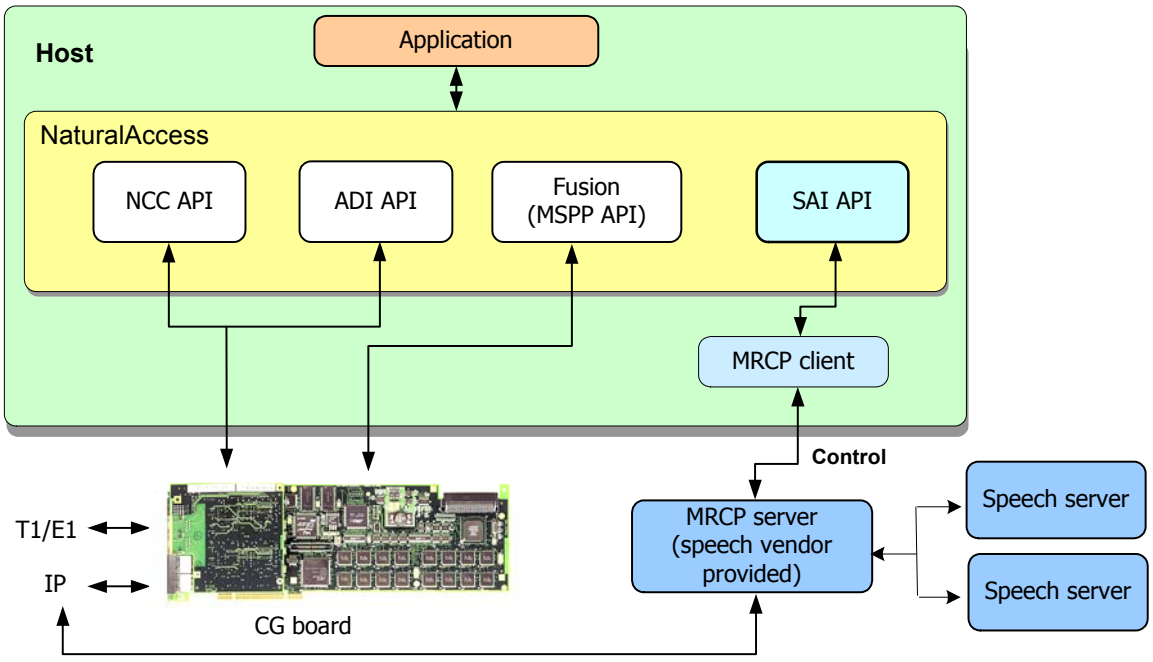
This API...	Provides...
ADI	DTMF detection and tone generation
NCC	PSTN call control
MSPP	RTP endpoint control
USAI	Universal Speech Access API speech recognition and speech synthesis

Example

The following example shows how the application processes a PSTN call and requests speech resources with USAI in the NaturalAccess development environment:

Step	Action
1	The telephony gateway accepts the call (using the NCC API) and connects the PSTN channel to a local stream.
2	The application requests a speech resource (ASR or TTS) from an MRCP server using USAI functions **saiCreateRecognizer** or **saiCreateSynthesizer**. When the speech resource is created, the MRCP server returns the created speech resource ID and the voice over IP (VoIP) port it uses to receive and transmit data.
3	The telephony gateway receives the information, creates an RTP endpoint (using MSPP API functions), and connects the endpoint to the call.
4	The application manages the speech resource with USAI functions. For example, the application can perform speech recognition or synthesis tasks, add or modify grammars, or get recognition results.

The following illustration provides an overview of the Universal Speech Access API architecture:



NaturalAccess environment

This topic presents:

- NaturalAccess programming model
- NaturalAccess Fusion API
- NaturalAccess OAM API

NaturalAccess programming model

NaturalAccess employs an asynchronous programming model to take advantage of concurrent processing. When called, most functions return immediately indicating the operation was initiated. The application can then perform other functions while NaturalAccess is processing the command.

There are two types of functions in NaturalAccess:

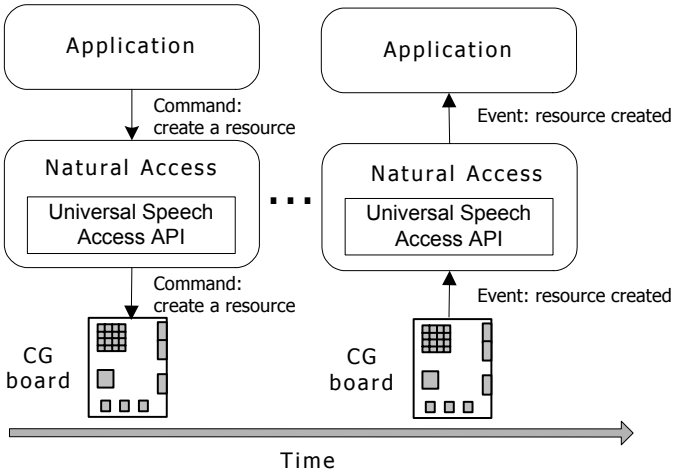
Function type	Description
Synchronous	Complete when the return value is received. The return value may be either SUCCESS or an error code.
Asynchronous	Returns events that indicate the success or failure of the operation. If the operation fails, it may return the reason for this failure in the value field of the event.

During the function execution, events are generated indicating the occurrence of certain conditions or state changes. If an asynchronous function fails after being initiated, NaturalAccess delivers a DONE event to the application and the event value field contains an error code.

The following table summarizes the differences between asynchronous and synchronous functions. The Function summary section in this manual lists the USAI functions and indicates if they are synchronous or asynchronous.

Characteristic	Asynchronous	Synchronous
Operation complete when function returns	No	Yes
NaturalAccess returns a DONE event when function is complete	Yes	No
Function can fail after function returns	Yes	No

As shown in the following illustration, for asynchronous functions, NaturalAccess sends a command to the API, which sends a command to a telephony board. The board performs the requested functions and sends events to the service indicating its state (for example, the function was started or the function is complete). The API sends events to NaturalAccess, which makes them available to the application. The following illustration shows NaturalAccess and the Universal Speech Access API interaction:



For detailed information about NaturalAccess, refer to the *Dialogic® NaturalAccess™ Software Developer's Manual*.

NaturalAccess Fusion API

NaturalAccess Fusion API is a voice-over-IP media streaming development environment. The MSPP API enables applications to establish connections by creating media streaming endpoints and channels. The service provides a standard set of endpoint and channel types with which to build these connections. For more information, refer to the *Dialogic® NaturalAccess™ Fusion™ VoIP API Developer's Manual.*

NaturalAccess OAM API

NaturalAccess OAM API manages and maintains telephony resources in a system, including:

- Hardware components, such as CG boards.
- Software components, such as the Hot Swap process.

Using the NaturalAccess OAM API, you can:

- Create, delete, and query the configuration of a component.
- Start (boot), stop (shut down), and test a component.
- Receive notifications from components.

Software processes supported by NaturalAccess OAM API include:

- Hot Swap: the ability to insert and extract CompactPCI boards without powering down the system.
- Clock management: the ability to configure the CT bus clock across all boards in an H.110 system.

For more information about NaturalAccess OAM API, refer to the *Dialogic® NaturalAccess™ OAM System Developer's Manual*.

Note: NaturalAccess OAM API operates only when the NaturalAccess Server (*ctdaemon*) is running.

Setting USAI parameters

Before initializing the Universal Speech Access API, specify configuration parameters in the USAI *saiconfig.xml* configuration file, located in the following directories:

- Windows: *c:\NMS\CTAccess\CFG\saiconfig.xml*
- UNIX: */opt/nms/ctaccess/cfg/saiconfig.xml*

Note: You must use a text editor to edit configuration entries. No comments are permitted in the file.

Universal Speech Access API configuration file parameters include:

Open tag	Parameter	Close tag	Description
<SpeechServer	ID=***servername***		Specifies the name of a speech server on which to create synthesizer or recognizer resources.
	Server=***speechvendor***		Specifies the MRCP speech server vendor. Currently not implemented.
	Deploy=false|true	>	Specifies whether or not the speech server is deployed. Valid entries include: • true: The MRCP client can attempt to connect to the server. • false: The MRCP client will not try to connect to the server.

Open tag	Parameter	Close tag	Description
<Protocol>	MRCP/***version***	</Protocol>	Specifies the media resource control protocol version used to communicate with the MRCP server. For example: MRCP/1.0 Currently only MRCP/1.0 is supported.
<Session>	***protocol***/***version***	</Session>	Specifies the session control protocol used to communicate with the MRCP server. For example: RTSP/1.0 Currently only RTSP/1.0 is supported.
<Transport>	***transportype***	</Transport>	Specifies the transport protocol used to communicate with the MRCP server. Currently only TCP is supported.
<Address>	***IPaddress***/***RTSPport***	</Address>	***IPaddress*** specifies the MRCP server IP address or the hostname. ***RTSPport*** specifies the RTSP port used when sending RTSP messages to the MRCP server (that is, the port on which the server listens for requests).
<Share>	***false|true***	</share>	OPTIONAL (default = false):Specifies whether or not TTS and ASR will use the same SessionID. (i.e. ASR address/port == TTS address/port)
<ASR>	<Enable> ***false|true***	</Enable>	OPTIONAL (default = true):If the MRCP server is ASR capable, this turns on/off ASR APIs.
	<ResourceName>***Recognizer***	</ResourceName>	OPTIONAL (default = recognizer):Recognizer defines the ASR resource name assigned by the MRCP server.
		</ASR>	Ends ASR parameters.
<TTS>	<Enable> ***false|true***	</Enable>	OPTIONAL (default = true):If the MRCP server is TTS capable, this turns on/off TTS APIs.
	<ResourceName>***synthesizer***	</ResourceName>	OPTIONAL (default = synthesizer):synthesizer defines the TTS resource name assigned by the MRCP server.
		</TTS>	Ends TTS parameters.
		</SpeechServer>	Ends USAI parameters.

The following example shows sample configuration file entries. SPEECHSERVER1 sets the optional parameters and SPEECHSERVER2 uses the default:

```
<SpeechServerConfiguration >
<SpeechServer ID="SPEECHSERVER1" Server="Generic" Deploy="false">
<Protocol>MRCP/1.0</Protocol>
<Session>RTSP/1.0</Session>
<Transport>TCP</Transport>
<Address>192.168.0.1:554</Address>
<Share>false</Share>
<ASR>
<Enable>true</Enable>
<ResourceName>recognizer</ResourceName>
</ASR>
<TTS>
<Enable>true</Enable>
<ResourceName>synthesizer</ResourceName>
</TTS>
</SpeechServer>
<SpeechServer ID="SPEECHSERVER2" Server="Generic"  Deploy="false">
<Protocol>MRCP/1.0</Protocol>
<Session>RTSP/1.0</Session>
<Transport>TCP</Transport>
<Address>HostnameOfSpeechServer2:4900</Address>
</SpeechServer>
</SpeechServerConfiguration>
```

5 Developing applications

Initializing USAI

Before calling functions from the USAI, the application initializes NaturalAccess and opens the required NaturalAccess APIs, including USAI.

To set up NaturalAccess:

Step	Action
1	Initialize NaturalAccess and its services.
2	Create event queues and contexts (associated with existing event queues).
3	Opening Universal Speech Access API instances on contexts.

Initializing NaturalAccess

To initialize NaturalAccess and its services, invoke **ctaInitialize** and specify the names of the services and service managers the application will use.

The following table lists some of the APIs and service managers that Universal Speech Access applications can use:

This API...	With this service manager...	Provides...
ADI	ADIMGR	DTMF detection and tone generation
NCC	ADIMGR	PSTN call control
MSPP	MSPMGR	RTP endpoint control
USAI	SAIMGR	Universal Speech Access API speech recognition and speech synthesis

For a list of the standard NaturalAccess APIs, refer to the *Dialogic® NaturalAccess™ Software Developer's Manual*.

Creating event queues and contexts

After initializing NaturalAccess and its services, the application creates event queues and contexts.

Step	Action
1	Create one or more event queues by invoking **ctaCreateQueue**. Specify which service managers to attach to each queue. USAI uses the SAIMGR service manager.
2	Create contexts by invoking **ctaCreateContext** and providing a queue handle (***ctaqueuehd***) returned from a previous call to **ctaCreateQueue**. Events for any services opened on the context are later routed to the associated event queue.

ctaCreateContext returns a context handle (***ctahd***). The application must specify a valid ***ctahd*** when invoking USAI functions. Events communicated back to the application are associated with the specified context handle. For information about

NaturalAccess programming models and managing contexts and queues, refer to the *Dialogic® NaturalAccess™ Software Developer's Manual*.

Opening Universal Speech Access API instances

Opening an API on a context creates a service instance on that context. To open a USAI instance on a context, invoke **ctaOpenServices** and specify:

- A ***ctahd***.
- The type of API (for example, USAI) to open.
- The appropriate service manager (for example, SAIMGR) for the specified API.

Creating synthesizer resources

After initializing the Universal Speech Access API, applications use USAI functions to request text-to-speech (TTS) synthesizer resources from a speech server. Each synthesizer resource is associated with a specific NaturalAccess context.

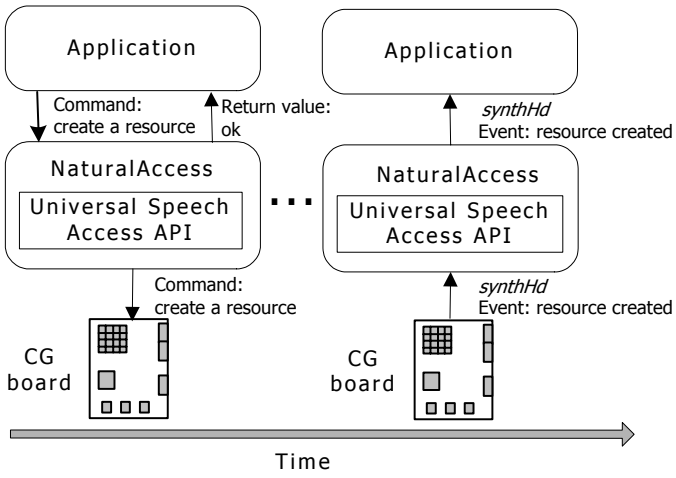
Applications invoke **saiCreateSynthesizer** to allocate synthesizer resources. When using **saiCreateSynthesizer**, the application must specify the following:

- NaturalAccess context handle.
- Incoming Fusion RTP endpoint information, including the IP address and the port number to listen on.
- (Optional) Name of a speech server where synthesizer requests are directed. If no speech server is specified, USAI defaults to the first server defined in the *saiconfig.xml* file.
- (Optional) Client object identifier that USAI associates with any events returned for the synthesizer resource.
- Pointer to a synthesizer handle container to receive the created synthesizer handle.

saiCreateSynthesizer is an asynchronous function. When **saiCreateSynthesizer** executes successfully, indicating that all specified parameters are valid, USAI sends a setup command to the speech server requesting a new synthesizer resource according to the specified parameters. USAI returns a synthesizer handle (***synthHd***) that the application references in subsequent requests for that resource. However, the application must wait for USAI to return SAIEVN_ENGINE_READY for the specified synthesizer before making any requests with the returned ***synthHd***.

Note: After an application invokes **saiCreateSynthesizer**, the only command that USAI can execute before receiving SAIEVN_ENGINE_READY is **saiReleaseSynthesizer**.

The following illustration shows NaturalAccess and USAI interaction when creating synthesizer resources:



The following example shows how an application creates a USAI synthesizer resource:

```
DWORD rc = SUCCESS;
CTAQUEUEHD queueHd;
CTAHD contextHd;
SAI_SYNTHESIZER_HANDLE synthHd;

CTA_SERVICE_NAME cta_srvc_name;
CTA_INIT_PARMS cta_init_parms;
CTA_SERVICE_DESC cta_srvc_desc;
SAI_URL g_server = "SpeechServer";
SAI_RTP_ENDPOINT rtpEndpoint;
CTA_EVENT *event;
DWORD serverPortTts = 0;

// Initializing Natural Access environment
ctaInitialize(cta_srvc_name, cta_init_parm);
ctaCreateQueue(NULL, 0, queueHd);
ctaCreateContext(queueHd, 0, NULL, contextHd);
ctaOpenServices(contextHd, cta_srvc_desc, 1);
rtpEndPoint.Port = 2000;
rtpEndPoint.ipAddress = "10.10.1.1";

    rc = saiCreateSynthesizer(contextHd, rtpEndpoint, g_server, 0, synthHd );
    if ( rc == SUCCESS )
    {
        // Wait for SAIEVN_ENGINE_READY or SAIEVN_ENGINE_FAIL
        rc = ctaWaitEvent( queueHd, event, 0 );
        if ( (rc == SUCCESS) && (event == SAIEVN_ENGINE_READY))
        {
            SAI_EVENT_ENGINE_READY* serverPort = (SAI_EVENT_ENGINE_READY*) (event->buffer
);
            printf("Engine created successfuly, Port # :%d\n", serverPortàenginePort);
        }
            else if((rc == SUCCESS) && (event == SAIEVN_ENGINE_FAIL))
        {
            printf("Engine creation failed, engine not ready.\n");
        }
    }
    else
    {
        printf("saiCreateSynthesizer failed.\n");
    }
```

Creating recognizer resources

After initializing the Universal Speech Access API, applications use USAI functions to request automatic speech recognition (ASR) resources from a speech server. Each recognizer resource is associated with a specific NaturalAccess context.

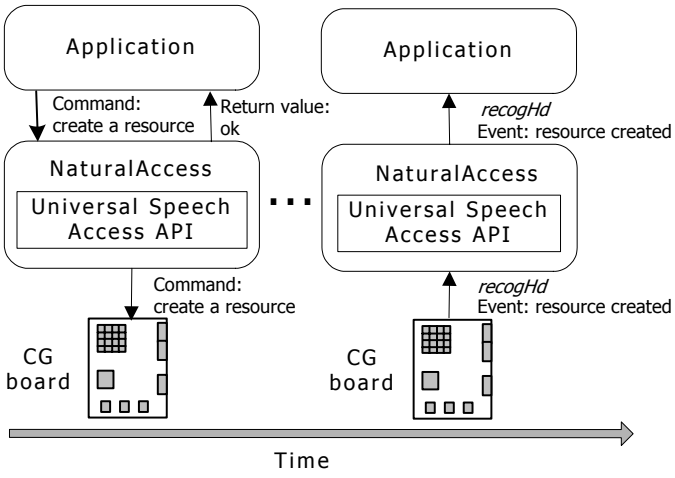
Applications invoke **saiCreateRecognizer** to allocate recognizer resources. When using **saiCreateRecognizer**, the application must specify the following:

- NaturalAccess context handle.
- Incoming Fusion RTP endpoint information, including the IP address and the port number to transmit over.
- (Optional) Name of a speech server where recognizer requests are directed. If no speech server is specified, USAI defaults to the first server defined in the *saiconfig.xml* file.
- (Optional) Client object identifier that USAI associates with any events returned for the recognizer resource.
- Pointer to a recognizer handle container to receive the created recognizer handle.

saiCreateRecognizer is an asynchronous function. When **saiCreateRecognizer** completes successfully (indicating that all specified parameters are valid), USAI sends a setup command to the speech server requesting a new recognizer resource according to the specified parameters. USAI returns a recognizer handle (***recogHd***) that the application references in subsequent requests to the resource. However, the application must wait for USAI to return SAIEVN_ENGINE_READY for the specified recognizer before making any requests with the returned ***recogHd***.

Note: After an application invokes **saiCreateRecognizer**, the only command that USAI can execute before the return of SAIEVN_ENGINE_READY is **saiReleaseRecognizer**.

The following illustration shows NaturalAccess and Universal Speech Access API interaction when creating recognizer resources:



The following example shows how an application creates an Universal Speech Access API recognizer resource:

```
DWORD rc = SUCCESS;
CTAQUEUEHD queueHd;
CTAHD contextHd;
SAI_RECOGNIZER_HANDLE recogHd;
CTA_SERVICE_NAME cta_srvc_name;
CTA_INIT_PARMS cta_init_parms;
CTA_SERVICE_DESC cta_srvc_desc;
SAI_URL g_server  = "SpeechServer";
SAI_RTP_ENDPOINT rtpEndpoint;
CTA_EVENT *event;

// Initialize Natural Access environment
ctaInitialize(cta_srvc_name,cta_init_parm)
ctaCreateQueue(NULL, 0, queueHd)
ctaCreateContext(queueHd, 0, NULL, contextHd)
ctaOpenServices(contextHd, cta_srvc_desc, 1)
rtpEndPoint.Port = 2000;
rtpEndPoint.ipAddress = "10.10.1.1"

     rc = saiCreateRecognizer(contextHd, rtpEndpoint, g_server, 0, recogHd );
     if ( rc == SUCCESS )
     {
          // Wait for SAIEVN_ENGINE_READY or SAIEVN_ENGINE_FAIL
          rc = ctaWaitEvent( queueHd, event, 0 );
          if ( (rc == SUCCESS) && (event == SAIEVN_ENGINE_READY))
          {
               SAI_EVENT_ENGINE_READY* serverPort = (SAI_EVENT_ENGINE_READY*) (event-
>buffer );
               printf("Engine created successfuly, Port # :%d\n", serverPortàenginePort);
          }
          else if((rc == SUCCESS) && (event == SAIEVN_ENGINE_FAIL))
          {
               printf("Engine creation failed, engine not ready.\n");
          }
     }
     else
     {
          printf("saiCreateRecognizer failed.\n");
```

Creating a synthesizer speak list

Applications define synthesizer speak lists that the server uses when the application sends multiple sentences of text to a synthesizer at one time. A speak list contains entries of a single data type or of different data types.

When the speak list contains different data types, the synthesizer uses multiple-part speak mode in accordance with MRCP 7.5.1 to perform speech synthesis. For information about multiple-part speak mode support for a specific synthesizer, refer to the speech vendor documentation.

Applications invoke **saiInitSpeakList**, **saiAddSpeakItem**, and **saiSpeakSynthesizer** when requesting speech synthesis one sentence at a time.

The following example shows how an application sends multiple sentences of text in various forms to a synthesizer:

```
#define SAI_TEST_MAX_GRAMMAR_LIST_SIZE      0x1000
char speakRuleList[SAI_TEST_MAX_GRAMMAR_LIST_SIZE];
char textToSpeak[] = "This is the SAI demo program";
char xmlToSpeak[] =
"<?xml version="1.0"?>"
" <speak>\n "
"  <paragraph>\n"
"    <sentence>You have 4 new messages.</sentence>\n"
"    <sentence>The first is from <say-as>\n"
"    type="name">Stephanie Williams</say-as>\n"
"    and arrived at <break/>\n"
"    <say-as type="time">3:45pm</say-as>.</sentence>\n"
"    <sentence>The subject is <prosody>\n"
"    rate="-20%">ski trip</prosody></sentence>\n"
"  </paragraph>\n";
" </speak>\n"
char uriToSpeak[] = http://www.nmss.com/TTS-Introduction.sml;

     rc = saiInitSpeakList(speakRuleList, SAI_TEST_MAX_GRAMMAR_LIST_SIZE) );
    if ( rc == SUCCESS )
    {
        //
        // Add speak rules to the datalist.
        //
        rc = saiAddSpeakItem(speakRuleList, SAI_SPEAK_DATA_TYPE_PLAIN_TEXT, textToSpeak);
        rc = saiAddSpeakItem(speakRuleList, SAI_SPEAK_DATA_TYPE_XML, xmlToSpeak);
        rc = saiAddSpeakItem(speakRuleList, SAI_SPEAK_DATA_TYPE_URL, uriToSpeak);
        rc = saiSpeakSynthesizer( synthHd, SAI_SPEAK_DATA_TYPE_DATA_LIST, speakRuleList,
        NULL );
    }
```

Creating a recognition grammar

A typical speech recognizer uses static and dynamic grammars to recognize speech input and to match results. Static grammars are loaded at start up and are always available. Dynamic grammars are loaded for specific recognition sessions and are associated with a particular recognition session (and therefore, a specific ***recogHd***).

Before streaming voice data to a recognizer, the application can define a grammar list that the recognizer can load and use to recognize speech input and match results. The application uses **saiInitRecognitionList, saiAddRecognitionGrammar**, and **saiDefineGrammarRecognizer** to send the grammar list to the recognizer. This provides a dynamic grammar that the recognizer can use for the entire recognizer session.

After the application invokes **saiDefineGrammarRecognizer**, the speech server attempts to compile the specified grammars and returns SAIEVN_ASR_DEFINE_GRAMMAR_DONE. The completion event indicates whether or not the grammar list was successfully compiled. If compilation fails, the cause of the failure is specified in the completionCause field of the SAIEVN_ASR_DEFINE_GRAMMAR_DONE event buffer.

Grammar lists can contain grammar rules in various formats, including XML formatted text or simple URLs. After the speech server compiles a grammar list, it saves the grammar in a server-side database. The speech server uses the grammar list whenever the application requests a speech recognition task until the recognizer resource is released with **saiReleaseRecognizer**. Each time the application invokes **saiStartRecognizer**, it specifies a pre-loaded grammar for the server to use to carry out the specified recognition task.

Note: Recognition grammar databases are only valid for a specific recognition session. Once the session is closed, the grammar list is not valid for other recognition sessions.

The following example shows how an application creates a recognition grammar:

```
DWORD grmListSize = 0x1000;
void *grmList = calloc( 1, grmListSize );
char dtmf_digit_grammar[] =
"<?xml version=\"1.0\"?>\n"
"<grammar xmlns=\"http://www.w3.org/2001/06/grammar\"\n"
" xml:lang=\"en-US\" version=\"1.0\" mode=\"dtmf\"\n"
" root=\"four-digits\">\n"
"\n"
" <rule id=\"digit\">\n"
" <one-of>\n"
" <item> 1 </item>\n"
" <item> 2 </item>\n"
" <item> 3 </item>\n"
" <item> 4 </item>\n"
" <item> 5 </item>\n"
" <item> 6 </item>\n"
" <item> 7 </item>\n"
" <item> 8 </item>\n"
" <item> 9 </item>\n"
" <item> 0 </item>\n"
" </one-of>\n"
" </rule>\n"
" <rule id=\"four-digits\" scope=\"public\">\n"
" <item>\n"
" <ruleref uri=\"#digit\"/>\n"
" <ruleref uri=\"#digit\"/>\n"
" <ruleref uri=\"#digit\"/>\n"
" <ruleref uri=\"#digit\"/>\n"
" </item>\n"
" </rule>\n"
"</grammar>\n";

    rc = saiInitRecognitionList( grmList, grmListSize );
    if (rc == SUCCESS)
    {
    rc = saiAddRecognitionGrammar(gramlist,SAI_GRAMMAR_TYPE_XML,NULL,dtmf_digit_grammar);
    rc = saiDefineGrammarRecognizer (recogHd, gramlist, NULL );
    }
```

Managing USAI parameters

The Universal Speech Access API provides three types of parameter values:

- System defaults
- Per-session
- Per-command

System default parameter values

System default parameter values are set at speech server start up time. These parameter values are usually defined in speech server configuration files and are vendor-specific. To modify these values, use vendor-specific configuration methods.

Per-session parameter values

Session default parameters maintain specific parameter values as long as a current synthesizer or recognizer session remains open. The initial settings for these values are the system defaults. To modify these values on the client side, use USAI synthesizer and recognizer parameter functions such as **saiAsrSetSpeedVsAccuracy** and **saiTtsSetVoice**. Refer to *Recognizer parameter functions* on page 36 and *Synthesizer parameter functions* on page 39.

The application can call multiple synthesizer and recognizer parameter functions to modify parameters on the client side. However, these parameter modifications are not updated on the speech server until the application uses **saiSetParamRecognizer** or **saiSetParamSynthesizer**. By invoking **saiSetParamRecognizer** and **saiSetParamSynthesizer**, the application updates the specified parameter modifications on the speech server, and the modified parameters remain valid for the duration of the speech server session. All subsequent commands associated with the current synthesizer or recognizer session use the modified parameter values.

If the application invokes synthesizer and recognizer parameter functions but does not invoke **saiSetParamRecognizer** or **saiSetParamSynthesizer**, the modified parameters only affect the next synthesizer or recognizer command that the application invokes.

Per-command parameter values

Per-command parameter settings maintain specified values until the application sends a synthesizer or recognizer request to the server. To modify synthesizer or recognizer parameter values on the client side, use USAI synthesizer and recognizer parameter functions such as **saiAsrSetSpeedVsAccuracy** and **saiTtsSetVoice**. Refer to *Recognizer parameter functions* on page 36 and *Synthesizer parameter functions* on page 39.

However, when the application calls a USAI recognizer or synthesizer function other than **saiSetParamRecognizer** or **saiSetParamSynthesizer** (for example, **saiSpeakSynthesizer**), USAI only applies the modified parameter values for that command. After the command executes, all modified parameters revert to the session defaults.

Example of managing recognizer parameters

The following table shows how SAI service parameter values are applied when an application uses Universal Speech Access API recognizer functions:

Task	Client-side action	Client-side parameter value	Server-side action	Server-side parameter value
Server start up.	No action.	System default	Server initialization. Read configuration file and update server-side database.	System default
Client start up.	Requests server-side default parameter values and updates the client-side database.	System default	No action.	System default
Start a speech session and request a recognizer resource.	Creates a recognizer object and the system default value is applied to the current session.	System default	Allocates a recognizer resource to the current session. The system default values are applied.	System default
Request a recognition action.	Application calls **saiStartRecognizer**. Client sends a recognize request to the server with no new parameter values.	System default	Performs the recognition action and returns the result.	System default
Retrieve server parameter value for speed versus accuracy.	Application calls **saiAsrGetSpeedVsAccuracy**. Client returns value from system default parameter values.	System default	No action.	System default
Set client-side parameter value for speed versus accuracy.	Application calls **saiAsrSetSpeedVsAccuracy**. Client updates its local database parameter value.	Per-command	No action.	System default
Update server-side parameter values with client-side settings.	Application calls **saiSetParamRecognizer**. Client sends a SET_PARAM request to the server and specifies the new parameter value.	Per-session	Updates the parameter for the current session only.	Per-session
Retrieve server-side parameter values.	Application calls **saiGetParamRecognizer**. Client sends a GET_PARAM request to the server and retrieves the parameter value.	Per-session	Updates client-side values.	Per-session
Recognition complete.	Reverts to the previous parameter value before the call to **saiStartRecognizer**.	System default	Reverts to the previous parameter value before the **saiStartRecognizer** command.	System default

Task	Client-side action	Client-side parameter value	Server-side action	Server-side parameter value
Retrieve server parameter value for speed versus accuracy.	Application calls **saiAsrGetSpeedVsAccuracy**. Client returns value from system default parameter values.	System default	No action.	System default
Set client-side parameter value for speed versus accuracy.	Application calls **saiAsrSetSpeedVsAccuracy**. Client updates its local database parameter value.	Per-command	No action.	System default
Update server-side parameter values with client-side settings	Application calls **saiSetParamRecognizer**. Client sends a SET_PARAM request to the server and specifies the new parameter value.	Per-session	Updates the parameter for the current session only.	Per-session
Request a recognition action.	Application calls **saiStartRecognizer**. Client sends a recognize request to the server.	Per-session	Performs the recognition action according to the current per-session parameters and returns the result.	Per-session
Request a recognition action.	Application calls **saiStartRecognizer**. Client sends a recognize request to the server.	Per-session	Performs the recognition action according to the current per-session parameters and returns the result.	Per-session
Close the session.	Application calls **saiReleaseRecognizer**. Client sends a release request to the server and closes the recognizer object.	System default	Frees the recognizer resource.	System default

Implementing a barge-in enabled prompt

Universal Speech Access API applications use barge-in detection to interrupt an active voice prompt when a caller begins speaking before the prompt finishes. USAI enables applications to implement barge-in detection in the following ways:

- Synchronizing the actions of a recognizer and a synthesizer that run in different sessions. USAI notifies the synthesizer through SAIEVN_START_OF_SPEECH. When the recognizer detects speech input, the synthesizer stops generating its speech prompt.
- Using a vendor-specific link between the synthesizer and recognizer.

This topic describes two scenarios for implementing a barge-in enabled prompt:

- Performing barge-in with recognizers and synthesizers in different sessions
- Performing barge-in with linked recognizers and synthesizers

Performing barge-in with recognizers and synthesizers in different sessions

When speech resources are not on the same session and the recognizer and synthesizer processes are not linked, the MRCP client can act as a proxy by sending start-of-speech messages received from a recognizer to a specific synthesizer. This process works in the following way:

Step	Action
1	The recognizer detects the beginning of speech input.
2	USAI returns SAIEVN_START_OF_SPEECH to the client application.
3	The client application uses **saiNotifyBargeInToSynthesizer** to send a BARGE-IN-OCCURRED event to the synthesizer that indicates that the recognizer resource has detected speech input.
4	The synthesizer receives a BARGE-IN-OCCURRED event and stops playing the prompt
5	The recognizer begins recognizing the input voice data.

Configuring recognizer and synthesizer parameters

To implement a barge-in enabled prompt, the application must start and configure the recognizer and synthesizer in the following way:

- Create and configure the recognizer so that the recognizer detects speech input before the synthesizer begins to play a voice prompt.
- Start the synthesizer with barge-in enabled.

To implement kill-on-barge-in for unlinked synthesizers and recognizers:

Step	Action
1	The application creates a recognizer resource with **saiCreateRecognizer**, after making sure (with **saiAsrSetRecognitionStartTimer**) that all timers are disabled.
2	The application creates a synthesizer resource with **saiCreateSynthesizer**.
3	The application enables barge-in for the synthesizer with **saiTtsSetKillOnBargeIn** (setting the value to TRUE).
4	The application plays a voice prompt with **saiSpeakSynthesizer**.
5	If the recognizer detects voice input before the synthesizer finishes generating its voice prompt, the recognizer sends a START-OF-SPEECH message to USAI.
6	USAI returns SAIEVN_START_OF_SPEECH to the application.
7	The application associates the received SAIEVN_START_OF_SPEECH with a particular synthesizer session and invokes **saiNotifyBargeInToSynthesizer** to notify the synthesizer that a barge-in has occurred.
8	The synthesizer automatically stops playing the active voice prompt (the application does not have to call **saiStopSynthesizer**) and eliminates all voice prompts currently in its queue.
9	The application invokes **saiStartTimerRecognizer** to start the recognizer's no-input-timeout timer. The no-input-timeout timer specifies the length of time to wait during which no voice input is detected before ending the recognizer task.

Note: saiAsrSetRecognitionStartTimer specifies a TRUE or FALSE value that enables or disables recognizer timers (the recognition timeout as well as the no-input-timeout). **saiAsrSetRecognitionTimeout** specifies the maximum length of time that the recognizer waits before terminating a recognition request when speech is detected.

Performing barge-in with linked recognizers and synthesizers

If the synthesizer or recognizer vendor supports linked recognizer and synthesizer tasks, you can interrupt an active synthesizer prompt more quickly than if it is managed by the application and USAI.

To implement kill-on-barge-in for linked synthesizers and recognizers:

Step	Action
1	The application creates a recognizer resource with **saiCreateRecognizer**, with its no-input-timeout timer disabled (this is the default configuration).
2	The application creates a synthesizer resource with **saiCreateSynthesizer**.
3	The application enables barge-in for the synthesizer with **saiTtsSetKillOnBargeIn** (setting the value to TRUE).
4	The application plays a voice prompt with **saiSpeakSynthesizer**.
5	If the recognizer detects voice input before the synthesizer finishes generating its voice prompt, the recognizer sends a START-OF-SPEECH message to USAI.
6	USAI returns SAIEVN_START_OF_SPEECH to the application.
7	The application associates SAIEVN_START_OF_SPEECH with a particular synthesizer session and invokes **saiNotifyBargeInToSynthesizer** to notify the synthesizer that a barge-in has occurred. The recognizer sends a stop command to the synthesizer before USAI returns SAIEVN_START_OF_SPEECH to the client application. Even if the synthesizer is already stopped, the MRCP client application must close the loop. A ProxySyncId identifier is embedded into the SAIEVN_START_OF_SPEECH event returned to the application. The application must specify this ProxySyncId when invoking **saiNotifyBargeInToSynthesizer**.

6 Function summary

Recognizer functions

The Universal Speech Access API provides the following functions for controlling recognizer operations:

Function	Synchronous/ Asynchronous	Description
saiAddRecognitionGrammar	Synchronous	Adds a grammar definition to the grammar set. The definition is used when defining a grammar and starting a recognizer.
saiCreateRecognizer	Asynchronous	Creates a recognizer instance and allocates recognizer engine resources.
saiDefineGrammarRecognizer	Asynchronous	Defines the grammar set used for a speech recognition process.
saiGetParamRecognizer	Asynchronous	Requests recognizer parameters from the speech server and updates the associated client-side recognizer resource information.
saiGetResultRecognizer	Asynchronous	Requests the recognizer engine to recompute the recognition result based on parameters set with **saiAsrSetConfidenceThreshold** and **saiAsrSetSensitivity**.
saiInitRecognitionList	Synchronous	Initializes the grammar list used by **saiAddRecognitionGrammar**, **saiDefineGrammarRecognizer**, and **saiStartRecognizer**.
saiReleaseRecognizer	Synchronous	Releases a recognizer instance and de-allocates recognizer engine resources.
saiSetParamRecognizer	Asynchronous	Sets the recognizer parameters used until the specified recognizer is released.
saiStartRecognizer	Asynchronous	Activates a recognition process.
saiStartTimerRecognizer	Asynchronous	Activates a no-input timer for a recognizer engine.
saiStopRecognizer	Asynchronous	Stops a speech recognition process.

Recognizer parameter functions

The Universal Speech Access API provides the following synchronous functions for retrieving and setting recognizer parameters:

Function	Description
saiAsrGetConfidenceThreshold	Retrieves the confidence level the recognizer uses to determine a successful match.
saiAsrSetConfidenceThreshold	Sets the confidence level the recognizer uses to determine a successful match.
saiAsrGetDtmfInterDigitTimeout	Retrieves the inter-digit timeout value used when recognizing DTMF input.
saiAsrSetDtmfInterDigitTimeout	Sets the inter-digit timeout value used when recognizing DTMF input.
saiAsrGetDtmfTermChar	Retrieves the terminating DTMF character for DTMF input recognition.
saiAsrSetDtmfTermChar	Sets the terminating DTMF character for DTMF input recognition.
saiAsrGetDtmfTermTimeout	Retrieves the terminating timeout interval to use when recognizing DTMF input.
saiAsrSetDtmfTermTimeout	Sets the terminating timeout interval to use when recognizing DTMF input.
saiAsrGetFetchTimeout	Retrieves the timeout interval the recognizer waits before fetching the resources that the media server needs from the network when the recognizer needs grammar documents.
saiAsrSetFetchTimeout	Sets the timeout interval the recognizer waits before fetching the resources that the media server needs from the network when the recognizer needs grammar documents.
saiAsrGetNBestListLength	Retrieves the maximum number of alternative results the recognizer can return for matches above the confidence threshold.
saiAsrSetNBestListLength	Sets the maximum number of alternative results the recognizer returns for matches above the confidence threshold.
saiAsrGetNewAudioChannel	Retrieves whether or not the client can request that the media server send audio data from a new audio source.
saiAsrSetNewAudioChannel	Sets whether or not the client can request that the media server send audio data from a new audio source.
saiAsrGetNoInputTimeout	Retrieves the no-input-timeout interval that the recognizer waits before completing the recognition request.
saiAsrSetNoInputTimeout	Sets the no-input-timeout interval that the recognizer waits before completing the recognition request.
saiAsrGetRecognitionStartTimer	Retrieves whether or not the recognizer must start the no-input timer at the beginning of the recognizer session.
saiAsrSetRecognitionStartTimer	Sets whether or not the recognizer must start the no-input timer at the beginning of the recognizer session.

Function	Description
saiAsrGetRecognitionTimeout	Retrieves the recognition-timeout interval that the recognizer waits before completing the recognition request.
saiAsrSetRecognitionTimeout	Sets the recognition-timeout interval that the recognizer waits before completing the recognition request.
saiAsrGetSaveWaveform	Retrieves whether or not the recognizer must save recognized audio streams.
saiAsrSetSaveWaveform	Sets whether or not the recognizer must save recognized audio streams.
saiAsrGetSensitivity	Retrieves the recognizer sensitivity level.
saiAsrSetSensitivity	Defines the recognizer sensitivity level.
saiAsrGetSpeechCompleteTimeout	Retrieves the interval that the recognizer waits before declaring a match when there is a complete match of an active grammar.
saiAsrSetSpeechCompleteTimeout	Sets the interval that the recognizer waits before declaring a match when there is a complete match of an active grammar.
saiAsrGetSpeechIncompleteTimeout	Retrieves the interval that the recognizer waits before declaring a match when there is an incomplete match of all active grammars.
saiAsrSetSpeechIncompleteTimeout	Sets the interval that the recognizer waits before declaring a match when there is an incomplete match of all active grammars.
saiAsrGetSpeechLanguage	Retrieves the recognition grammar language for data within a session or request.
saiAsrSetSpeechLanguage	Specifies the recognition grammar language for data within a session or request.
saiAsrGetSpeedVsAccuracy	Retrieves the recognizer speed versus accuracy setting.
saiAsrSetSpeedVsAccuracy	Sets the recognizer speed versus accuracy setting.
saiAsrAddVendorSpecific	Adds vendor-specific recognizer parameters to the local database.
saiAsrGetVendorSpecific	Retrieves vendor-specific recognizer parameters from the local database.
saiAsrSetVendorSpecific	Sets vendor-specific recognizer parameters in the local database.

Synthesizer functions

The Universal Speech Access API provides the following functions for controlling synthesizer operations:

Function	Synchronous/ Asynchronous	Description
saiAddSpeakItem	Asynchronous	Adds a specified speak rule to the end of the specified speak list.
saiControlSynthesizer	Asynchronous	Controls a synthesizer process.
saiCreateSynthesizer	Asynchronous	Creates a synthesizer instance and allocates synthesizer engine resources.
saiGetParamSynthesizer	Asynchronous	Requests synthesizer parameters from the speech server and updates the associated client-side synthesizer resource information.
saiInitSpeakList	Asynchronous	Initializes the speak list buffer used by **saiAddSpeakItem** and **saiSpeakSynthesizer**.
saiNotifyBargeInToSynthesizer	Asynchronous	Notifies the synthesizer that a barge-in occurred.
saiPauseSynthesizer	Asynchronous	Pauses a synthesizer process.
saiReleaseSynthesizer	Synchronous	Releases a synthesizer instance and de-allocates synthesizer engine resources.
saiResumeSynthesizer	Asynchronous	Resumes a paused synthesizer process.
saiSetParamSynthesizer	Synchronous	Updates the synthesizer parameters used for the duration of the current session.
saiSpeakSynthesizer	Asynchronous	Starts a speech synthesis process.
saiStopSynthesizer	Asynchronous	Stops a speech synthesis process.

Synthesizer parameter functions

The Universal Speech Access API provides the following synchronous functions for retrieving and setting synthesizer parameters:

Function	Description
saiTtsGetAudioFetchHint	Retrieves whether or not the synthesizer can optimize speech by pre-fetching audio.
saiTtsSetAudioFetchHint	Specifies whether or not the synthesizer can optimize speech by pre-fetching audio.
saiTtsGetFetchHint	Retrieves when the synthesizer retrieves content from a document server.
saiTtsSetFetchHint	Specifies when the synthesizer retrieves content from a document server.
saiTtsGetFetchTimeout	Retrieves the synthesizer timeout period for waiting for resources from the network.
saiTtsSetFetchTimeout	Specifies the synthesizer timeout period for waiting for resources from the network.
saiTtsGetKillOnBargeIn	Retrieves whether or not kill-on-barge-in support is enabled for a specified synthesizer.
saiTtsSetKillOnBargeIn	Enables or disables kill-on-barge-in support for a specified synthesizer.
saiTtsGetProsody	Retrieves synthesizer speech prosody parameters.
saiTtsSetProsody	Specifies synthesizer speech prosody parameters.
saiTtsGetSpeakerProfile	Retrieves the URL where the speaker profile is kept.
saiTtsSetSpeakerProfile	Sets the URL where the profile of the speaker is kept.
saiTtsGetSpeechLanguage	Retrieves the default language used when synthesizing speech.
saiTtsSetSpeechLanguage	Specifies the default language used when synthesizing speech.
saiTtsAddVendorSpecific	Adds vendor-specific synthesizer parameters to the local database.
saiTtsGetVendorSpecific	Retrieves vendor-specific synthesizer parameters from the local database.
saiTtsSetVendorSpecific	Specifies vendor-specific synthesizer parameters for the local database.
saiTtsGetVoice	Retrieves speaker voice characteristics based on SSML specifications.
saiTtsSetVoice	Specifies speaker voice characteristics based on SSML specifications.

7 Recognizer function reference

Using the recognizer function reference

This section provides an alphabetical reference to the recognizer functions in the Universal Speech Access API. A typical function includes:

Prototype	The prototype is followed by a list of the function arguments. Data types include: • WORD (16-bit unsigned) • DWORD (32-bit unsigned) • INT16 (16-bit signed) • INT32 (32-bit signed) • BYTE (8-bit unsigned) If a function argument is a data structure, the complete data structure is defined. Refer to *Universal Speech Access API structures* on page 163 for more information.
Return values	The return value for a function is either SUCCESS or an error code. For asynchronous functions, a return value of SUCCESS indicates the function was initiated; subsequent events indicate the status of the operation. Refer to *Universal Speech Access API errors* on page 139 for more information.
Events	If events are listed, the function is asynchronous and is complete when the DONE event is returned. If there are no events listed, the function is synchronous. Additional information such as reason codes and return values may be provided in the value field of the event. Each event consists of an event ID, an event header value field, and the informational event buffer. Refer to *Universal Speech Access API events* on page 141 for more information.
Example	Example functions taken from sample application programs shipped with the product. The notation `/* ... */` indicates additional code that is not shown.

saiAddRecognitionGrammar

Adds a grammar definition to the grammar set. The definition is used when defining a grammar and starting a recognizer.

Prototype

DWORD **saiAddRecognitionGrammar** (void ***gramlist***, SAI_GRAMMAR_TYPE ***gramtype***, char ***gramname***, char ***grammar***)

Argument	Description
gramlist	Pointer to a grammar list to which to add the grammar definition.
gramtype	Grammar type of the recognition grammar definition (URL or XML).
gramname	Pointer to a grammar name associated with the grammar entry.
grammar	Pointer to a new added grammar of type SAI_GRAMMAR.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_LIST_NOT_INITIALIZED	Grammar list referenced by ***gramlist*** was not initialized with **saiInitRecognitionList**.
SAIERR_NOT_ENOUGH_MEMORY	Grammar container referenced by ***gramlist*** does not have enough space for the new grammar.

Details

Before calling **saiAddRecognitionGrammar**, applications must call **saiDefineGrammarRecognizer**. For more information, refer to Creating a recognition grammar.

Example

See **saiStartRecognizer**.

saiCreateRecognizer

Creates a recognizer instance and allocates recognizer engine resources.

Prototype

DWORD **saiCreateRecognizer** (CTAHD ***ctahd***, SAI_RTP_ENDPOINT ****rtpEndpoint***, char ****server***, DWORD ***clientObjId***, SAI_RECOGNIZER_HANDLE ****recogHd***)

Argument	Description
ctahd	Context handle associated with a call.

rtpEndpoint	Pointer to the SAI_RTP_ENDPOINT structure specifying a client RTP endpoint where the RTP stream originates:
	`typedef struct`
	`{`
	`DWORD port;`
	`SAI_URL ipAddress;`
	`}`
	`SAI_RTP_ENDPOINT;`
	For more information, refer to *SAI_RTP_ENDPOINT* on page 163.

server	Pointer to a URL associated with a speech server. This argument is optional. If the application specifies NULL, the speech engine controller uses the default speech server name.
clientObjId	(Optional). Application ***clientObjId*** to associate with the recognizer engine resource. All events associated with the new resource are associated with this ***clientObjId***.
recogHd	Pointer to a returned SAI service recognizer handle associated with a recognizer resource.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
CTAERR_INVALID_HANDLE	Specified ***ctahd*** is invalid.

Events

Event	Description
SAIEVN_ENGINE_CREATION_DONE	Command completed. The event statusCode field in the SAI_EVENT_ENGINE_CREATION event indicates the success or failure of the operation. If the command completed successfully (the status code is SC_CSC_OK), the application can use the returned ***recogHd***. If the operation failed, the application must call **saiReleaseRecognizer** to release the returned ***recogHd***.

Details

saiCreateRecognizer allocates a recognizer resource and associates it with a ***recogHd***. When the speech server successfully allocates the new recognizer resource, the recognizer enters a ready state and the application receives SAIEVN_ENGINE_READY. The RTP port on which the server listens is returned in the enginePort field.

Note: From the time the application creates the recognizer to the time that the server responds to the request, the recognizer remains in an in-progress state. In this state, all requests from the application to the recognizer are rejected.

For more information, refer to *Creating recognizer resources* on page 24.

See also

saiCreateSynthesizer

Example

```
//
// The ASR must be associated to a CTA context.
// The RTP endpoint defines the source where the client generates the audio stream.
// It defines the endpoint with IP address and port number.
// The port number must be even as specified by RTP protocol.
// The asrHd returned by that function is used to refer to that ASR engine for all
// recognition command.
//
// The Application waits on the event queue to receive the SAIEVN_ENGINE_CREATION_DONE.
// This event returns the server RTP endpoint. The RTP endpoint defines the destination
// where the server terminates the audio stream. It contains IP address and port number.
//
DWORD demoAsrCreate( CTAQUEUEHD queueHd, CTAHD ctxtHd, SAI_RECOGNIZER_HANDLE *asrHd,
                     SAI_RTP_ENDPOINT EpAsr, DWORD *serverPortAsr)
{
    DWORD rc = SUCCESS;
    CTA_EVENT event;
    SAI_EVENT_ENGINE_CREATION* eventResult = NULL;
    memset( &event, 0, sizeof (CTA_EVENT) );
    if (*asrHd != 0)
    {
        printf(" A ASR already exists.\n");
return !SUCCESS;
    }
    rc = saiCreateRecognizer( ctxtHd, EpAsr, g_server, 0, asrHd );
    if ( rc == SUCCESS )
    {
        //
        // Verify each event received in queue as long as the appropriate one
        // (in this case: SAIEVN_ENGINE_CREATION_DONE) is received or before timeout
        // is over.
        //
        rc = demoWaitForEvent( queueHd, &event, SAIEVN_ENGINE_CREATION_DONE, 0);
    }
    if ( rc == SUCCESS)
    {
//
// The buffer pointer contains a speech access event. Cast buffer according to the
        // event type.
//
eventResult = (SAI_EVENT_ENGINE_CREATION*) ( event.buffer );
//
// Verify that completion cause and status code are the same as buffer memory
//
if ( (eventResult != NULL) &&
     (eventResult->statusCode == SAI_SC_SUCCESS) )
{
    *serverPortAsr = eventResult->enginePort;
}
```

```
else
{
            printf("Status Code = 0x%08x.\n", eventResult->statusCode);
    rc = !SUCCESS;
}
    }
    if (rc == SUCCESS)
    {
        printf("Engine creation succeed on serverport = %d and on serverIp = %s.\n",
              *serverPortAsr, eventResult->engineIp);
    }
    else
    {
        printf("Engine creation failed, error code: 0x%08x.\n", rc);
    }
    return rc;
}
```

saiDefineGrammarRecognizer

Defines the grammar set used for a speech recognition process.

Prototype

DWORD **saiDefineGrammarRecognizer** (SAI_RECOGNIZER_HANDLE ***recogHd***, void ****grmList***, char ****grmName***, SAI_REQUEST_ID ****requestId***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
grmList	Pointer to a grammar list that was initialized and loaded.
grmName	Pointer to the grammar name.
requestId	Pointer to a unique request identifier associated with the request.

Return values

Return value	Description
SUCCESS	
SAIERR_ENGINE_NOT_READY	Specified recognizer engine is not ready to handle requests.
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_LIST_EMPTY	Grammar list contains no entries.
SAIERR_LIST_NOT_INITIALIZED	Grammar list referenced by ***grmList***was not previously initialized with **saiInitRecognitionList**.

Events

Event	Description
SAIEVN_ASR_DEFINE_GRAMMAR_DONE	Request was processed. The result of the request is returned in the statusCode field of the SAIEVN_ASR_DEFINE_GRAMMAR_DONE event buffer.

Details

saiDefineGrammarRecognizer defines a list of grammars to use during the next speech recognition process. If necessary, the server downloads and compiles the specified grammar.

If the recognizer resource is in the recognition state, it cannot respond to the request. The SAI service then indicates a request failure in the statusCode field of the SAIEVN_ASR_DEFINE_GRAMMAR_DONE event buffer, and indicates the cause for the failure in the completionCause field of the event buffer.

For more information, refer to *Creating a recognition grammar* on page 27.

See also

saiAddRecognitionGrammar

Example

See **saiStartRecognizer**.

saiGetParamRecognizer

Requests recognizer parameters from the speech server and updates the associated client-side recognizer resource information.

Prototype

DWORD **saiGetParamRecognizer** (SAI_RECOGNIZER_HANDLE ***recogHd***, SAI_REQUEST_ID ****requestId***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
requestId	Pointer to a unique identifier associated with the request.

Return values

Return value	Description
SUCCESS	
SAIERR_ENGINE_NOT_READY	Specified recognizer engine is not ready to handle requests.
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.

Events

Event	Description
SAIEVN_GET_PARAMETER_DONE	Request was processed. The result of the request is returned in the statusCode field of the SAIEVN_GET_PARAMETER_DONE event buffer.

Details

saiGetParamRecognizer retrieves parameter values from the server and updates the recognizer properties. Applications use **saiGetParamRecognizer** to synchronize client-side parameters with the server-side parameters.

When **SaiGetParamRecognizer** executes successfully, the application can retrieve recognizer parameters using the SAI service parameter function associated with the parameter (for example, **saiAsrGetConfidenceThreshold** or **saiAsrGetSensitivity**).

See also

saiSetParamRecognizer

saiGetResultRecognizer

Requests the recognizer engine to recompute the recognition result based on parameters set with **saiAsrSetConfidenceThreshold** and **saiAsrGetNBestListLength**.

Prototype

DWORD **saiGetResultRecognizer** (SAI_RECOGNIZER_HANDLE ***recogHd***, SAI_REQUEST_ID ****requestId***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
requestId	Pointer to a unique identifier associated with the request.

Return values

Return value	Description
SUCCESS	
SAIERR_ENGINE_NOT_READY	Specified recognizer engine is not ready to handle requests.
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.

Events

Event	Description
SAIEVN_ASR_GET_RESULT_DONE	Request was processed. The result of the request is returned in the statusCode field of the SAIEVN_ASR_GET_RESULT_DONE event buffer.

Details

If the recognizer does not support resetting recognition processes with new recognition parameters, the server rejects the command and the SAI service returns SAI_SC_METHOD_NOT_ALLOWED in the statusCode field of the SAIEVN_ASR_GET_RESULT_DONE event buffer.

See also

saiDefineGrammarRecognizer, **saiStartRecognizer**

saiInitRecognitionList

Initializes the grammar list used by **saiAddRecognitionGrammar**, **saiDefineGrammarRecognizer**, and **saiStartRecognizer**.

Prototype

DWORD **saiInitRecognitionList** (void ****grmList***, DWORD ***grmListSize***)

Argument	Description
grmList	Pointer to a grammar list.
grmListSize	Size of the grammar list.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_NOT_ENOUGH_MEMORY	Grammar list is not large enough for the recognizer to use. The grammar list must contain at least enough memory for the main list header and at least one header per grammar entry.

Details

The ***grmListSize*** value must include the internal list header information and the data size of each item to be added. Use the helper macro SAI_COMPUTE_LIST_OVERHEAD (***nElement***, ***overheadSize***) to calculate the total overhead size according to the number of items in the list. The grammar list requires at least one main header and one header per grammar entry.

Example

See **saiStartRecognizer**.

saiReleaseRecognizer

Releases a recognizer instance and de-allocates recognizer engine resources.

Prototype

DWORD **saiReleaseRecognizer** (SAI_RECOGNIZER_HANDLE ***recogHd***)

Argument	Description
recogHd	Handle associated with a recognizer resource.

Return values

Return value	Description
SUCCESS	
SAIERR_ENGINE_NOT_READY	Specified recognizer engine is not ready to handle requests.
SAIERR_INVALID_PARAMETER	***recogHd*** is invalid.

Details

saiReleaseRecognizer de-allocates recognizer resources associated with a specified speech server. If **saiReleaseRecognizer** executes successfully, the recognizer enters a deletion state and the SAI service deletes the recognizer object after the server responds. Until the server responds, the SAI service returns an SAIERR_ENGINE_NOT_READY error in response to any requests associated with the associated ***recogHd***.

If ***recogHd*** is not associated with a recognizer, the handle is not freed and the application receives SAIERR_INVALID_PARAMETER.

For more information, refer to *Creating recognizer resources* on page 24.

See also

saiCreateRecognizer

Example

```
// Destruction of ASR engine object using SAI function. Its handle is equal to 0.
// The application does not need to wait on the event queue to receive a specific event
// because SAI service releases all recognition engine trace.
//
DWORD demoAsrRelease( SAI_RECOGNIZER_HANDLE *recogHd, DWORD *serverPortAsr)

{
    DWORD rc = SUCCESS;

    rc = saiReleaseRecognizer(*asrHd );
    if ( rc == SUCCESS )
    {
         printf("Releasing ASR succeed.\n");
         *serverPortAsr = 0;
         *asrHd = 0;
    }
    else
    {
         printf("Failure on releasing ASR, error code = 0x%08x.\n", rc);
    }
   return rc;
}
```

saiSetParamRecognizer

Sets the recognizer parameters used until the specified recognizer is released.

Prototype

DWORD **saiSetParamRecognizer** (SAI_RECOGNIZER_HANDLE ***recogHd***, SAI_REQUEST_ID ****requestId***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
requestId	Pointer to a unique identifier associated with the request.

Return values

Return value	Description
SUCCESS	
SAIERR_ENGINE_NOT_READY	Specified recognizer engine is not ready to handle requests.
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.

Events

Event	Description
SAIEVN_SET_PARAMETER_DONE	Request was processed. The result of the request is returned in the statusCode field of the SAIEVN_SET_PARAMETER_DONE event buffer.

Details

Some SAI recognizer parameter functions such as **saiAsrSetConfidenceThreshold** and **saiAsrSetSensitivity** do not immediately alter the way the recognition server operates. Any modified parameters are stored locally (on the client side) until the application calls **saiSetParamRecognizer**. When the application invokes **saiSetParamRecognizer**, the accumulated parameter changes are sent to the speech server, and the changes are enforced for any subsequent recognition tasks until the recognizer is released. For more information, refer to *Managing USAI parameters* on page 29.

See also

saiGetParamRecognizer

saiStartRecognizer

Activates a recognition process.

Prototype

DWORD **saiStartRecognizer** (SAI_RECOGNIZER_HANDLE ***recogHd***, SAI_GRAMMAR_TYPE ***grmType***, void ****data***, SAI_REQUEST_ID ****requestId***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
grmType	List of pre-defined grammar sets. The following list shows string and decimal values for supported grammar types: • SAI_GRAMMAR_TYPE_URL [0]: The grammar includes a URL that is loaded by the recognizer engine. • SAI_GRAMMAR_TYPE_XML [1]: The grammar is formatted following XML rules. • SAI_GRAMMAR_TYPE_GRAMMAR_LIST [2]: The grammar is a list in table form.
data	Pointer to the grammar payload. This can be the grammar list previously defined.
requestId	Pointer to a unique request identifier returned for this command.

Return values

Return value	Description
SUCCESS	
SAIERR_ENGINE_NOT_READY	Specified recognizer engine is not ready to handle requests.
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_LIST_EMPTY	Grammar list referenced by ***data*** contains no entries.
SAIERR_LIST_NOT_INITIALIZED	Grammar list referenced by ***data*** was not initialized with **saiInitRecognitionList**.

Events

Event	Description
SAIEVN_ASR_RECOGNITION_DONE	Request is complete. The completion cause is provided in the completionCause field of the SAIEVN_ASR_RECOGNITION_DONE event buffer.
SAIEVN_ASR_RECOGNITION_STARTED	Recognizer start request is currently in process.
SAIEVN_START_OF_SPEECH	Unsolicited event indicating that the speech synthesizer process has begun.

Details

saiStartRecognizer requests that the recognizer begin performing speech recognition of a stream of voice data and provides the grammar to use (in the ***data*** argument) when performing this process. If the specified ***grmType*** is SAI_GRAMMAR_TYPE_GRAMMAR_LIST, ***data*** contains a previously compiled list of grammar names. When invoking **saiStartRecognizer**, the application can specify parameters that control the following aspects of recognizer processing:

- Sensitivity
- Confidence level
- Level of detail in results generated by the recognizer

Parameters that the application specifies when invoking **saiStartRecognizer** override parameter settings established by a previous call to **saiSetParamRecognizer**. However, the parameter values revert to their initial values when the SAI service returns SAIEVN_ASR_RECOGNITION_DONE.

For more information, refer to *Creating a recognition grammar* on page 27.

See also

saiAddRecognitionGrammar, **saiDefineGrammarRecognizer**, **saiGetResultRecognizer**, **saiStopRecognizer**

Example

```
//
// Start recognition with grammar list name. ASR handle is used to reach the appropriate
// recognition engine created previously.
// The Application waits on the event queue to receive the SAIEVN_ASR_RECOGNITION_STARTED
and
// SAIEVN_ASR_RECOGNITION_DONE.
//
// Grammars are defined on server and can now be reached by URL path during the current
speech
// session. With this function using grammar list names, grammar can actually be called
with its
// corresponding name tag (enumerated in a grammar list name) and attributed  when
calling
// saiAddRecognitionGrammar.
//
DWORD demoAsrStartGrmListName( CTAQUEUEHD queueHd, SAI_RECOGNIZER_HANDLE asrHd )
{
    DWORD rc = SUCCESS;
    SAI_REQUEST_ID reqId = 0;
    DWORD grmListSize = TEST_MAX_GRAMMAR_LIST_SIZE;
    void* grmList = calloc( 1, grmListSize );
    rc = saiInitRecognitionList( grmList, grmListSize );
    if (rc == SUCCESS)
    {
        rc = saiAddRecognitionGrammar(grmList, SAI_GRAMMAR_TYPE_XML, "grammar1",
             G_speech_digit_grammar_enUS );
    }
    if (rc == SUCCESS)
    {
        //
        // Upload the grammar for compilation on server
        //
        rc = saiDefineGrammarRecognizer( asrHd, grmList, "myDefinedGrammar", &reqId );
    }
    if ( rc == SUCCESS )
    {
        //
        // Verify each event received in queue as long as the appropriate one
        // (in this case: SAIEVN_ASR_DEFINE_GRAMMAR_DONE) are received or before
```

```
        // timeout is over. Verify that status code and reqId are the same as buffer
memory.
        //
        rc = demoWaitForEventDefGramFcn(queueHd, reqId);
    }
    if ( rc == SUCCESS )
    {
        //
        // Start recognition process using the grammar list that has been defined on
server
        // (so it becomes a recognition using URL type), reqId associated to the current
        // request is returned.
        //
        SAI_URL urlListName[] = { {"myDefinedGrammar"},
                                  {""}
                                };
rc = saiStartRecognizer(asrHd, SAI_GRAMMAR_TYPE_URL, urlListName, &reqId );
    }
    if ( rc == SUCCESS )
    {
        //
        // Verify each event received in queue as long as the appropriate one
        // (in this case: SAIEVN_ASR_RECOGNITION_STARTED and SAIEVN_ASR_RECOGNITION_DONE)
are
        // received or before timeout is over. Verify that status code, completion cause
and
        // reqId are the same as buffer memory.
        //
    rc = demoWaitForEventRecognitionFcn(queueHd, reqId);
    }
    if (rc == SUCCESS)
    {
        printf("Recognition with SAI_GRAMMAR_TYPE_URL(grammar list name) succeed!.\n");

    }
    else
    {
        printf("Recognition with SAI_GRAMMAR_TYPE_URL(grammar list name) failed, error
code = 0x%08X.\n", rc);
    }
    return rc;
}
```

saiStartTimerRecognizer

Activates the timers for a recognizer engine.

Prototype

DWORD **saiStartTimerRecognizer** (SAI_RECOGNIZER_HANDLE ***recogHd***, SAI_REQUEST_ID ****requestId***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
requestId	Pointer to a unique request identifier returned for this command.

Return values

Return value	Description
SUCCESS	
SAIERR_ENGINE_NOT_READY	Specified recognizer engine is not ready to handle requests.
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.

Events

Event	Description
SAIEVN_ASR_START_TIMER_DONE	Request was processed. The result of the request is returned in the statusCode field of the SAIEVN_ASR_START_TIMER_DONE event buffer.

Details

saiStartTimerRecognizer is used to implement a kill-on-barge-in enabled prompt. An IVR application starts the recognition timers when:

- The voice prompt finishes playing.
- Speech input is detected while the voice prompt is still active.

To detect a barge-in occurrence, the recognizer needs to be active, but the application usually does not start the recognizer's no-input timer and recognition timer until either the voice prompt finishes or barge-in occurs. In either case, applications then call **saiStartTimerRecognizer** to start the timers. For more information, refer to *Implementing a barge-in enabled prompt* on page 32.

Applications can invoke **saiAsrSetRecognitionStartTimer** to specify whether or not the timers are enabled when the application calls **saiStartRecognizer**. The application can also use **saiAsrSetNoInputTimeout** and **saiAsrSetRecognitionTimeout** to specify the length of the timer intervals. Other timers supported by a speech server behave in the same way.

saiStopRecognizer

Stops a speech recognition process.

Prototype

DWORD **saiStopRecognizer** (SAI_RECOGNIZER_HANDLE ***recogHd***, SAI_REQUEST_ID ****requestId***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
requestId	Pointer to a unique request identifier returned for this command.

Return values

Return value	Description
SUCCESS	
SAIERR_ENGINE_NOT_READY	Specified recognizer engine is not ready to handle requests.
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.

Events

Event	Description
SAIEVN_ASR_STOP_DONE	Request was processed. The result of the request is returned in the statusCode field of the SAIEVN_ASR_STOP_DONE event buffer.

Details

saiStopRecognizer terminates a recognition process.

If the command interrupts an active recognition request and the statusCode field of the SAIEVN_ASR_STOP_DONE event buffer indicates SUCCESS, the stoppedReqId field specifies the requestId of the terminated request. When **saiStopRecognizer** interrupts an inactive recognition request, the SAI service does not return SAIEVN_ASR_STOP_DONE for the terminated request.

See also

saiStartRecognizer

8 Recognizer parameter function reference

Using the recognizer parameter function reference

This section provides an alphabetical reference to the recognizer parameter functions in the Universal Speech Access API . A typical function includes:

Prototype	The prototype is followed by a list of the function arguments. Data types include: • WORD (16-bit unsigned) • DWORD (32-bit unsigned) • INT16 (16-bit signed) • INT32 (32-bit signed) • BYTE (8-bit unsigned) If a function argument is a data structure, the complete data structure is defined. Refer to *Universal Speech Access API structures* on page 163 for more information.
Return values	The return value for a function is either SUCCESS or an error code. For asynchronous functions, a return value of SUCCESS indicates the function was initiated; subsequent events indicate the status of the operation. Refer to *Universal Speech Access API errors* on page 139 for more information.
Events	If events are listed, the function is asynchronous and is complete when the DONE event is returned. If there are no events listed, the function is synchronous. Additional information such as reason codes and return values may be provided in the value field of the event. Each event consists of an event ID, an event header value field, and the informational event buffer. Refer to *Universal Speech Access API events* on page 141 for more information.
Example	Example functions taken from sample application programs shipped with the product. The notation `/* ... */` indicates additional code that is not shown.

saiAsrAddVendorSpecific

Adds vendor-specific recognizer parameters to the local database.

Prototype

DWORD **saiAsrAddVendorSpecific** (SAI_RECOGNIZER_HANDLE ***recogHd***, SAI_VENDOR_SPECIFIC ***vp***[], DWORD ***count***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
vp	Pointer to an array of vendor-specific parameters to add to the local database. Only the vp[x].vendorSpecificName parameter must be specified.
count	Number of vendor-specific parameters to add. This value must not exceed the size of the ***vp***[].

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.

Details

Before retrieving or setting vendor-specific parameters, the application must first add them to the local database with **saiAsrAddVendorSpecific**. These settings are not transferred to the speech server until the application calls **saiSetParamRecognizer**.

See also

saiAsrGetVendorSpecific, **saiAsrSetVendorSpecific**

saiAsrGetConfidenceThreshold

Retrieves the confidence level the recognizer uses to determine a successful word-match.

Prototype

DWORD **saiAsrGetConfidenceThreshold** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ****value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Pointer to confidence level expressed as a percentage value between 0 - 100.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	The recognizer engine does not support the specified parameter.

Details

When a recognition resource recognizes or matches a spoken phrase with some portion of the grammar, it associates a confidence level with that conclusion. If the recognizer determines that its confidence in its recognition results is less than the confidence threshold, the SAI service returns SAI_RCC_NO_MATCH in the completionCause field of the SAIEVN_ASR_RECOGNITION_DONE event buffer.

See also

saiAsrSetConfidenceThreshold

saiAsrGetDtmfInterDigitTimeout

Retrieves the length of time that the recognizer waits between DTMF tones before ending the recognizer session.

Prototype

DWORD **saiAsrGetDtmfInterDigitTimeout** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ****value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Pointer to the timeout interval size in ms.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	The recognizer engine does not support the specified parameter.

See also

saiAsrSetDtmfInterDigitTimeout

saiAsrGetDtmfTermChar

Retrieves the DTMF character that signals the end of the recognizer session.

Prototype

DWORD **saiAsrGetDtmfTermChar** (SAI_RECOGNIZER_HANDLE ***recogHd***, SAI_DTMF_TERM_CHAR ***termChar***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
termChar	Pointer to a DTMF character.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	The recognizer engine does not support the specified parameter.

See also

saiAsrSetDtmfTermChar

saiAsrGetDtmfTermTimeout

Retrieves the length of time that the recognizer waits for DTMF input before ending the recognizer session.

Prototype

DWORD **saiAsrGetDtmfTermTimeout** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ****value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Pointer to the timeout interval size in ms.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

See also

saiAsrSetDtmfTermTimeout

saiAsrGetFetchTimeout

Retrieves the length of time that the recognizer waits before fetching the network resources for the media server when the recognizer needs grammar documents.

Prototype

DWORD **saiAsrGetFetchTimeout** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ****value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Pointer to the timeout interval size in ms.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

See also

saiAsrSetFetchTimeout

saiAsrGetNBestListLength

Retrieves the maximum number of alternative results the recognizer can return for matches above the confidence threshold.

Prototype

DWORD **saiAsrGetNBestListLength** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ****value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Pointer to the number of results to return.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

Details

When the recognizer matches an incoming stream with the specified grammar, it sometimes finds more than one match due to the word confidence levels or conversation paths. The best-list-length parameter sets the number of alternate results the recognizer returns in the SAIEVN_ASR_RECOGNITION_DONE event buffer.

See also

saiAsrGetConfidenceThreshold, **saiAsrSetNBestListLength**

saiAsrGetNewAudioChannel

Retrieves whether or not the client can request that the media server obtain audio data from a new audio source.

Prototype

DWORD **saiAsrGetNewAudioChannel** (SAI_RECOGNIZER_HANDLE ***recogHd***, BOOL ****value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Pointer to one of the following values: • TRUE: Client can request audio data from an alternate source. • FALSE: Client cannot request audio data from an alternate source.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

Details

When **saiAsrGetNewAudioChannel** is set to TRUE, the recognizer must discard all collected information or line statistics and start fresh for the new recognizer request.

Applications can call **saiAsrGetNewAudioChannel** to reuse an open recognition session on the media server for multiple telephone calls. This function enables applications to avoid any delay associated with creating a new recognizer and specifying new configuration parameters for that recognizer.

See also

saiAsrSetNewAudioChannel

saiAsrGetNoInputTimeout

Retrieves the length of time (no-input-timeout interval) that the recognizer waits before terminating a recognition request when no speech is detected.

Prototype

DWORD **saiAsrGetNoInputTimeout** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ****value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Pointer to a no-input timeout interval value in ms. The timeout interval value can range from 0 to MAXTIMEOUT, where MAXTIMEOUT is a recognizer-specific value. For information about supported values, refer to the speech vendor documentation.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

Details

When recognition is started and no speech is detected for a particular interval, the recognizer sends SAIEVN_ASR_RECOGNITION_DONE to the client and terminates the recognition operation. The SAI service returns SAI_RCC_INPUT_TIMEOUT in the completionCause field of the SAIEVN_ASR_RECOGNITION_DONE event buffer.

See also

saiAsrSetNoInputTimeout

saiAsrGetRecognitionStartTimer

Retrieves whether the recognizer starts its no-input timer at the beginning of the recognizer session or waits for a call to **saiStartTimerRecognizer**. For more information, refer to *Implementing a barge-in enabled prompt* on page 32.

Prototype

DWORD **saiAsrGetRecognitionStartTimer** (SAI_RECOGNIZER_HANDLE ***recogHd***, BOOL ****value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Pointer to one of the following values: • TRUE: Recognizer starts its no-input timer when the application calls **saiStartRecognizer**. • FALSE: Recognizer starts its no-input timer when the application calls **saiStartTimerRecognizer**.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

See also

saiAsrSetRecognitionStartTimer

saiAsrGetRecognitionTimeout

Retrieves the length of time (recognition-timeout interval) that the recognizer waits before terminating a recognition request when speech is detected.

Prototype

DWORD **saiAsrGetRecognitionTimeout** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ****value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Pointer to a timeout interval value in ms. The timeout interval value can range from 0 to MAXTIMEOUT, where MAXTIMEOUT is a recognizer-specific value. For information about supported values, refer to the speech vendor documentation.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

Details

When recognition starts and there is no match for a particular period of time, the recognizer sends SAIEVN_ASR_RECOGNITION_DONE to the application and terminates the recognition operation. The SAI service returns SAI_RCC_RECOGNITION_TIMEOUT in the completionCause field of the SAIEVN_ASR_RECOGNITION_DONE event buffer.

See also

saiAsrSetRecognitionTimeout

saiAsrGetSaveWaveform

Retrieves whether or not the recognizer must save recognized audio streams.

Prototype

DWORD **saiAsrGetSaveWaveform** (SAI_RECOGNIZER_HANDLE ***recogHd***, BOOL ****value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Pointer to one of the following values: • TRUE: Recognizer saves audio streams. • FALSE: Recognizer does not save audio streams.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

Details

When **saiAsrGetSaveWaveform** is set to TRUE, the recognizer must record the recognized audio and make it available to the client at the URL specified in the WaveformUrl field of the SAIEVN_ASR_RECOGNITION_DONE event buffer.

If an error occurs in recording the stream or the audio clip is not available, the recognizer must return an empty WaveformUrl field.

See also

saiAsrSetSaveWaveform

saiAsrGetSensitivity

Retrieves the recognizer sensitivity level.

Prototype

DWORD **saiAsrGetSensitivity** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ****value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Pointer to the recognizer sensitivity level. For information about the supported value range, refer to the speech vendor documentation.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

Details

Use **saiAsrGetSensitivity** to filter background noise from speech input. Some recognizers support a variable level of sound sensitivity. For information, refer to the speech vendor documentation.

See also

saiAsrSetSensitivity

saiAsrGetSpeechCompleteTimeout

Retrieves the length of time that the recognizer waits before declaring a match when there is a complete match of an active grammar.

Prototype

DWORD **saiAsrGetSpeechCompleteTimeout** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ****value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Pointer to an interval size in ms.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

Details

saiAsrGetSpeechCompleteTimeout retrieves the length of the interval of silence that a recognizer waits before finalizing a result and either accepting the result or returning a no-match event.

See also

saiAsrSetSpeechCompleteTimeout

saiAsrGetSpeechIncompleteTimeout

Retrieves the interval that the recognizer waits before declaring a match when there is an incomplete match of all active grammars.

Prototype

DWORD **saiAsrGetSpeechIncompleteTimeout** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ****value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Pointer to the timeout interval size in ms.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

Details

saiAsrGetSpeechIncompleteTimeout retrieves the length of the interval of silence that a recognizer waits before finalizing a result when the speech prior to the silence is an incomplete match of all active grammars. The interval also applies when the speech prior to the silence is a complete match of an active grammar, but where it is possible to speak further and still match the grammar.

See also

saiAsrSetSpeechIncompleteTimeout

saiAsrGetSpeechLanguage

Retrieves the language of recognition grammar data within a session or request when the language is not specified within the data (XML document).

Prototype

DWORD **saiAsrGetSpeechLanguage** (SAI_RECOGNIZER_HANDLE ***recogHd***, SAI_SPEECH_LANGUAGE ****language***, SAI_SPEECH_LANGUAGE_STRING ***speechLanguageString***)

Argument	Description
recogHd	Handle associated with a recognizer resource.

language	Pointer to the following structure:
	`typedef struct`
	`{`
	`SAI_SPEECH_LANGUAGE language;`
	`SAI_SPEECH_LANGUAGE_STRING languageString;`
	`}`
	`SAI_SPEECH_LANGUAGE;`

speechLanguageString	Pointer to a string that specifies the language to use. For a list of valid strings, refer to *SAI_SPEECH_LANGUAGE* on page 164.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

See also

saiAsrSetSpeechLanguage

saiAsrGetSpeedVsAccuracy

Retrieves the recognizer speed versus accuracy setting.

Prototype

DWORD **saiAsrGetSpeedVsAccuracy** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ****value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Pointer to the speed versus accuracy level. For information about the supported value range for the recognizer, refer to the speech vendor documentation.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

Details

Some recognizers allow applications to adjust the speed versus accuracy setting in the direction of greater performance or greater accuracy. Higher accuracy can entail more processing requirements and higher CPU utilization that can result in a lower calls capacity for the media server. For more information, refer to *Managing USAI parameters* on page 29 and to the speech vendor documentation.

See also

saiAsrSetSpeedVsAccuracy

saiAsrGetVendorSpecific

Retrieves a set of vendor-specific recognizer parameters from the local database.

Prototype

DWORD **saiAsrGetVendorSpecific** (SAI_RECOGNIZER_HANDLE ***recogHd***, SAI_VENDOR_SPECIFIC ***vp***[], DWORD ***count***, SAI_VENDOR_SPECIFIC_OUT ***vpOut***[])

Argument	Description
recogHd	Handle associated with a recognizer resource.
vp	Array of vendor-specific parameters to retrieve from the local database. Only the vp[x].vendorSpecificName parameter must be specified.
count	Number of vendor-specific parameters to retrieve.
vpOut	Array that specifies where to store retrieved vendor-specific parameters.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_VENDOR_SPECIFIC_ERROR	At least one of the specified parameters was not previously added to the local database.

Details

Before retrieving vendor-specific parameters, the application must add them to the local database with **saiAsrAddVendorSpecific** and retrieve the specified values from the server with **saiGetParamRecognizer**.

Note: The ***vp***[] and ***vpOut***[] arrays must be the same size and the number of specified parameters must not exceed the size of ***vp***[] and ***vpOut***[].

See also

saiAsrSetVendorSpecific

saiAsrSetConfidenceThreshold

Specifies the confidence level the recognizer uses to determine a successful word-match.

Prototype

DWORD **saiAsrSetConfidenceThreshold** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ***value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Confidence level expressed as a percentage value between 0 - 100.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

Details

When a recognition resource recognizes or matches a spoken phrase with some portion of the grammar, it associates a confidence level with that conclusion. If the recognizer determines that its confidence in its recognition results is less than the confidence threshold, the SAI service returns SAI_RCC_NO_MATCH in the completionCause field of the SAIEVN_ASR_RECOGNITION_DONE event buffer.

See also

saiAsrGetConfidenceThreshold

saiAsrSetDtmfInterDigitTimeout

Specifies the length of time that the recognizer waits between DTMF tones before ending the recognizer session.

Prototype

DWORD **saiAsrSetDtmfInterDigitTimeout** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ***value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Timeout interval size in ms.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

See also

saiAsrGetDtmfInterDigitTimeout

saiAsrSetDtmfTermChar

Specifies the DTMF character that signals the end of the recognizer session.

Prototype

DWORD **saiAsrSetDtmfTermChar** (SAI_RECOGNIZER_HANDLE ***recogHd***, SAI_DTMF_TERM_CHAR ***termChar***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
termChar	DTMF character.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

See also

saiAsrGetDtmfTermChar

saiAsrSetDtmfTermTimeout

Specifies the length of time that the recognizer waits for DTMF input before ending the recognizer session.

Prototype

DWORD **saiAsrSetDtmfTermTimeout** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ***value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Timeout interval size in ms.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

See also

saiAsrGetDtmfTermTimeout

saiAsrSetFetchTimeout

Specifies the length of time that the recognizer waits before fetching the network resources for the media server when the recognizer needs grammar documents.

Prototype

DWORD **saiAsrSetFetchTimeout** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ***value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Timeout interval size in ms.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

See also

saiAsrGetFetchTimeout

saiAsrSetNBestListLength

Sets the maximum number of alternative results the recognizer can return for matches above the confidence threshold.

Prototype

DWORD **saiAsrSetNBestListLength** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ***value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Number of results to return.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

Details

Occasionally when the recognizer matches an incoming stream with the specified grammar, the recognizer finds more than one match due to the word confidence levels or conversation paths. **saiAsrSetNBestListLength** sets the number of alternate results the recognizer returns in the SAIEVN_ASR_RECOGNITION_DONE event buffer.

See also

saiAsrGetNBestListLength

saiAsrSetNewAudioChannel

Specifies whether or not the client can request that the media server obtain audio data from a new audio source.

Prototype

DWORD **saiAsrSetNewAudioChannel** (SAI_RECOGNIZER_HANDLE ***recogHd***, BOOL ***value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	One of the following values: • TRUE: Client can request audio data from an alternate source. • FALSE: Client cannot request audio data from an alternate source.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

Details

When **saiAsrSetNewAudioChannel** is set to TRUE, the recognizer must discard any collected information or line statistics and start fresh for the new recognizer request.

Applications can call **saiAsrGetNewAudioChannel** to reuse an open recognition session on the media server for multiple telephone calls. This function enables applications to avoid any delay associated with creating a new recognizer and specifying configuration parameters for that recognizer.

saiAsrSetNoInputTimeout

Specifies the length of time (no-input-timeout interval) that the recognizer waits before terminating a recognition request when no speech is detected.

Prototype

DWORD **saiAsrSetNoInputTimeout** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ***value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	No-input timeout interval size in ms. The value can range from 0 to MAXTIMEOUT, where MAXTIMEOUT is a recognizer-specific value. For information about supported values, refer to the speech vendor documentation.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

Details

When recognition starts and no speech is detected for a particular interval, the recognizer sends SAIEVN_ASR_RECOGNITION_DONE to the application and terminates the recognition operation. The SAI service returns SAI_RCC_INPUT_TIMEOUT in the completionCause field of the SAIEVN_ASR_RECOGNITION_DONE event buffer.

See also

saiAsrGetNoInputTimeout

saiAsrSetRecognitionStartTimer

Specifies whether the recognizer starts its no-input timer at the beginning of the recognizer session or waits for a call to **saiStartTimerRecognizer**. For more information, refer to *Implementing a barge-in enabled prompt* on page 32.

Prototype

DWORD **saiAsrSetRecognitionStartTimer** (SAI_RECOGNIZER_HANDLE ***recogHd***, BOOL ***value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	One of the following values: • TRUE: Recognizer starts its no-input timer when the application calls **saiStartRecognizer**. • FALSE: Recognizer starts its no-input timer when the application calls **saiStartTimerRecognizer**.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

See also

saiAsrGetRecognitionStartTimer

saiAsrSetRecognitionTimeout

Specifies the length of time (recognition-timeout interval) that the recognizer waits before terminating a recognition request when speech is detected.

Prototype

DWORD **saiAsrSetRecognitionTimeout** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ***value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Timeout interval size in ms. The timeout interval value can range from 0 to MAXTIMEOUT, where MAXTIMEOUT is a recognizer-specific value. For information about supported values, refer to the speech vendor documentation.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

Details

When recognition starts and there is no match for a particular period of time, the recognizer sends SAIEVN_ASR_RECOGNITION_DONE to the client and terminates the recognition operation. The SAI service returns SAI_RCC_RECOGNITION_TIMEOUT in the completionCause field of the SAIEVN_ASR_RECOGNITION_DONE event buffer.

See also

saiAsrGetRecognitionTimeout

saiAsrSetSaveWaveform

Specifies whether or not the recognizer must save recognized audio streams.

Prototype

DWORD **saiAsrSetSaveWaveform** (SAI_RECOGNIZER_HANDLE ***recogHd***, BOOL ***value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	One of the following values: • TRUE: Recognizer saves audio streams. • FALSE: Recognizer does not save audio streams.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

Details

When **saiAsrSetSaveWaveform** is set to TRUE, the recognizer must record the recognized audio and make it available to the client at the URL specified in the WaveformUrl field of the SAIEVN_ASR_RECOGNITION_DONE event buffer.

If an error occurs in recording the stream or the audio clip is not available, the recognizer must return an empty WaveformUrl field.

See also

saiAsrGetSaveWaveform

saiAsrSetSensitivity

Sets the recognizer sensitivity level.

Prototype

DWORD **saiAsrSetSensitivity** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ***value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Recognizer sensitivity level. For information about the supported value range, refer to the speech vendor documentation.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

Details

Use **saiAsrSetSensitivity** to filter background noise from speech input. Some recognizers support a variable level of sound sensitivity. For information, refer to the speech vendor documentation.

See also

saiAsrGetSensitivity

saiAsrSetSpeechCompleteTimeout

Specifies the length of time that the recognizer waits before declaring a match when there is a complete match of an active grammar.

Prototype

DWORD **saiAsrSetSpeechCompleteTimeout** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ***value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Timeout interval size in ms.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

Details

saiAsrSetSpeechCompleteTimeout specifies the length of the interval of silence that a recognizer waits before finalizing a result and either accepting the result or returning a no-match event.

See also

saiAsrGetSpeechCompleteTimeout

saiAsrSetSpeechIncompleteTimeout

Sets the interval that the recognizer waits before declaring a match when there is an incomplete match of all active grammars.

Prototype

DWORD **saiAsrSetSpeechIncompleteTimeout** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ***value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Timeout interval size in ms.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

Details

saiAsrSetSpeechIncompleteTimeout sets the length of the interval of silence that a recognizer waits before finalizing a result when the speech prior to the silence is an incomplete match of all active grammars. The interval also applies when the speech prior to the silence is a complete match of an active grammar, but where it is possible to speak further and still match the grammar.

See also

saiAsrGetSpeechIncompleteTimeout

saiAsrSetSpeechLanguage

Sets the language of the recognition grammar data within a session or request when the language is not specified within the data (XML document).

Prototype

DWORD **saiAsrSetSpeechLanguage** (SAI_RECOGNIZER_HANDLE ***recogHd***, SAI_SPEECH_LANGUAGE ***language***, SAI_SPEECH_LANGUAGE_STRING ***speechLanguageString***)

Argument	Description
recogHd	Handle associated with a recognizer resource.

language	Structure:
	`typedef struct`
	`{`
	`SAI_SPEECH_LANGUAGE language;`
	`SAI_SPEECH_LANGUAGE_STRING languageString;`
	`}`
	`SAI_SPEECH_LANGUAGE;`

speechLanguageString	String that specifies the language to use. For a list of valid strings, refer to *SAI_SPEECH_LANGUAGE* on page 164.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

See also

saiAsrGetSpeechLanguage

saiAsrSetSpeedVsAccuracy

Sets the recognizer speed versus accuracy setting.

Prototype

DWORD **saiAsrSetSpeedVsAccuracy** (SAI_RECOGNIZER_HANDLE ***recogHd***, DWORD ***value***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
value	Speed versus accuracy level. For information about the supported value range for the recognizer, refer to the speech vendor documentation.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Recognizer engine does not support the specified parameter.

Details

Some recognizers allow applications to adjust the speed versus accuracy setting in the direction of greater performance or greater accuracy. Higher accuracy can entail more processing requirements and higher CPU utilization that can result in a lower calls capacity for the media server. For more information, refer to Managing USAI parameters and to the speech vendor documentation.

See also

saiAsrGetSpeedVsAccuracy

saiAsrSetVendorSpecific

Specifies a set of vendor-specific recognizer parameters in the local database.

Prototype

DWORD **saiAsrSetVendorSpecific** (SAI_RECOGNIZER_HANDLE ***recogHd***, SAI_VENDOR_SPECIFIC ***vp***[], DWORD ***count***)

Argument	Description
recogHd	Handle associated with a recognizer resource.
vp	Array of vendor-specific parameters values to set in the local database.
count	Number of vendor specific parameters to specify. This value indicates number of entries in ***vp***[].

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_VENDOR_SPECIFIC_ERROR	At least one of the provided parameters was not previously added to the local database.

Details

Before setting vendor-specific parameters, the application must first add them to the local database with **saiAsrAddVendorSpecific** and retrieve the specified values from the server with **saiSetParamRecognizer**.

See also

saiAsrGetVendorSpecific

9 Synthesizer function reference

Using the synthesizer function reference

This section provides an alphabetical reference to the synthesizer functions in the SAI service. A typical function includes:

Prototype	The prototype is followed by a list of the function arguments. Data types include: • WORD (16-bit unsigned) • DWORD (32-bit unsigned) • INT16 (16-bit signed) • INT32 (32-bit signed) • BYTE (8-bit unsigned) If a function argument is a data structure, the complete data structure is defined. Refer to *Universal Speech Access API structures* on page 163 for more information.
Return values	The return value for a function is either SUCCESS or an error code. For asynchronous functions, a return value of SUCCESS indicates the function was initiated; subsequent events indicate the status of the operation. Refer to *Universal Speech Access API errors* on page 139 for more information.
Events	If events are listed, the function is asynchronous and is complete when the DONE event is returned. If there are no events listed, the function is synchronous. Additional information such as reason codes and return values may be provided in the value field of the event. Each event consists of an event ID, an event header value field, and the informational event buffer. Refer to *Universal Speech Access API events* on page 141 for more information.
Example	Example functions taken from sample application programs shipped with the product. The notation `/* ... */` indicates additional code that is not shown.

saiAddSpeakItem

Adds a specified speak rule to the end of a specified speak list.

Prototype

DWORD **saiAddSpeakItem** (void ****speakList***, SAI_SPEAK_DATA_TYPE ***dataType***, char ****speakItem***)

Argument	Description
speakList	Pointer to a speak list to which to add the speak rule.
dataType	Payload data type. The following list shows supported speak data types and their associated numeric values: • SAI_SPEAK_DATA_TYPE_URL [0]: Data in URL format. • SAI_SPEAK_DATA_TYPE_XML [1]: Data in XML format. • SAI_SPEAK_DATA_TYPE_PLAIN_TEXT [2]: Data in non-formatted plain text format. • SAI_SPEAK_DATA_TYPE_DATA_LIST [3]: Data is a list of speak rules.
speakItem	Pointer to a speak rule to add.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_LIST_NOT_INITIALIZED	Specified list pointer is not initialized.
SAIERR_NOT_ENOUGH_MEMORY	Specified list is not large enough to accommodate the specified speak rule.

Details

saiAddSpeakItem builds the speak list used by **saiSpeakSynthesizer**.

Applications must initialize the specified speak list (***speakList***) with **saiInitSpeakList** before invoking **saiAddSpeakItem**. If the specified ***speakList*** is not initialized, the SAI service returns SAIERR_LIST_NOT_INITIALIZED.

For more information, refer to Creating a synthesizer speak list.

saiControlSynthesizer

Controls a synthesizer process.

Prototype

DWORD **saiControlSynthesizer** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_REQUEST_ID ****requestId***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.
requestId	Pointer to a unique identifier associated with the request.

Return values

Return value	Description
SUCCESS	
SAIERR_ENGINE_NOT_READY	Specified synthesizer engine is not ready to handle the request.
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.

Events

Event	Description
SAIEVN_TTS_CONTROL_DONE	Request was processed. The result of the request is returned in the statusCode field of the SAIEVN_TTS_CONTROL_DONE event buffer.

Details

saiControlSynthesizer sends a message to the specified speech synthesizer engine to modify what it is currently being spoken. **saiControlSynthesizer** enables applications to jump forward and backward through the spoken text, changing speech rate and modifying other speech parameters for the active (IN-PROGRESS) speak request.

Prior to invoking **saiControlSynthesizer**, the application must set speech control parameters with one of the following functions:

This function...	Sets the...
saiTtsSetVoice	Voice parameters for the speech data.
saiTtsSetProsody	Prosody parameters for the speech data.

See also

saiPauseSynthesizer, **saiResumeSynthesizer**, **saiSpeakSynthesizer**, **saiStopSynthesizer**

saiCreateSynthesizer

Creates a synthesizer instance and allocates synthesizer engine resources.

Prototype

DWORD **saiCreateSynthesizer** (CTAHD ***ctahd***, SAI_RTP_ENDPOINT ***rtpEndpoint***, char ****server***, DWORD ***clientObjId***, SAI_SYNTHESIZER_HANDLE ****synthHd***)

Argument	Description
ctahd	NaturalAccess context handle associated with a call.

rtpEndpoint	SAI_RTP_ENDPOINT structure that specifies a client RTP endpoint where the RTP stream terminates:
	`typedef struct`
	`{`
	`DWORD port;`
	`SAI_URL ipAddress;`
	`}`
	`SAI_RTP_ENDPOINT;`
	For more information, refer to *SAI_RTP_ENDPOINT* on page 163.

server	Pointer to the server ID as set in the configuration file. If set to null, the SAI service uses the first speech server name listed in the SAI configuration file.
clientObjId	Client object ID to associate with the created synthesizer engine. This parameter is optional.
synthHd	Pointer to a returned SAI service handle associated with a synthesizer resource.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified parameters is invalid.
CTAERR_INVALID_HANDLE	Specified ***ctahd*** parameter is invalid and therefore the returned ***synthHd*** is undefined.

Events

Event	Description
SAIEVN_ENGINE_CREATION_DONE	Command completed. The event statusCode field in the SAI_EVENT_ENGINE_CREATION event indicates the success or failure of the operation. If the command completed successfully (the status code is SC_CSC_OK), the application can use the returned ***synthHd***. If the operation failed, the application must call **saiReleaseSynthesizer** to release the returned ***synthHd***.

Details

saiCreateSynthesizer allocates a speech server synthesizer resource and associates it with a ***synthHd***. When the speech server successfully allocates the new synthesizer engine resource, the synthesizer enters a ready state and the application receives SAIEVN_ENGINE_READY. The RTP port through which the server transmits the synthesized voice stream is specified in the enginePort field.

Note: From the time the application creates the synthesizer to the time that the server responds to the request, requests from application to the synthesizer are rejected because the synthesizer is in an invalid state.

For more information, refer to *Creating synthesizer resources* on page 22.

See also

saiCreateRecognizer

Example

```
//
// The TTS must be associated to a CTA context.
// The RTP endpoint defines the destination where the client receives the audio stream.
// It defines the endpoint with IP address and port number.
// The port number must be even as specified by RTP protocol.
// The ttsHd returned by that function is used to refer to that TTS engine for all
synthesis
// command.
//
// The Application waits on the event queue to receive the SAIEVN_ENGINE_CREATION_DONE.
// This event returns the server RTP endpoint. The RTP endpoint defines the source
// where the server terminates the audio stream. It contains IP address and port number.
//
DWORD demoTtsCreate( CTAQUEUEHD queueHd, CTAHD ctxtHd, SAI_SYNTHESIZER_HANDLE *ttsHd,
                   SAI_RTP_ENDPOINT ttsEp, DWORD *serverPortTts )
{
    DWORD rc = SUCCESS;
    CTA_EVENT event;
    SAI_EVENT_ENGINE_CREATION* eventResult = NULL;
    memset( &event, 0, sizeof(CTA_EVENT) );
    if (*ttsHd != 0)
    {
        printf("A TTS already exists.\n");
        return !SUCCESS;
    }
    rc = saiCreateSynthesizer( ctxtHd, ttsEp, g_server, 0, ttsHd );
    if ( rc == SUCCESS )
    {
        //
        // Verify each event received in queue as long as the appropriate one
        // (in this case: SAIEVN_ENGINE_CREATION_DONE) is received or before timeout
       // is over.
        //
        rc = demoWaitForEvent( queueHd, &event, SAIEVN_ENGINE_CREATION_DONE, 0);
    }
    if ( rc == SUCCESS)
    {
        //
        // The buffer pointer contains a speech access event. Cast buffer according to
the
        // event type.
        //
        eventResult = (SAI_EVENT_ENGINE_CREATION*) ( event.buffer );
        //
        // Verify that completion cause and status code are the same as buffer memory
        //
        if ( (eventResult != NULL) &&
             (eventResult->statusCode == SAI_SC_SUCCESS) )
{
```

```
             *serverPortTts = eventResult->enginePort;
        }
        else
        {
            printf("Status Code = 0x%08x.\n", eventResult->statusCode);
            rc = !SUCCESS;
        }
    }
    if (rc == SUCCESS)
    {

        printf("Engine creation succeed on serverport = %d and on serverIp = %s.\n",
                *serverPortTts, eventResult->engineIp );
    }
    else
    {
        printf("Engine creation failed, error code: 0x%08x.\n", rc);
    }

    return rc;
}
```

saiGetParamSynthesizer

Requests synthesizer parameters from the speech server and updates the associated client-side synthesizer resource information.

Prototype

DWORD **saiGetParamSynthesizer** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_REQUEST_ID ****requestId***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.
requestId	Pointer to a unique identifier associated with the request.

Return values

Return value	Description
SUCCESS	
SAIERR_ENGINE_NOT_READY	Specified synthesizer engine is not ready to handle the request.
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.

Events

Event	Description
SAIEVN_COMMAND_ACK	Request was processed. The result of the request is returned in the statusCode field of the SAIEVN_COMMAND_ACK event buffer.

Details

When the application receives SAIEVN_GET_PARAMETER_DONE, it can retrieve parameters using functions associated with each synthesizer parameter (for example, **saiTtsGetKillOnBargeIn** or **saiTtsGetSpeakerProfile**).

See also

saiSetParamSynthesizer

saiInitSpeakList

Initializes the speak list buffer used by **saiAddSpeakItem** and **saiSpeakSynthesizer**.

Prototype

DWORD **saiInitSpeakList** (void ***speakList***, DWORD ***speakListSize***)

Argument	Description
speakList	Pointer to a new speak list buffer.
speakListSize	Size of the speak list.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_NOT_ENOUGH_MEMORY	Speak list is not big enough to be used. It should contain at least enough memory for the main list header and at least one item header from a speak list.

Details

The ***speakListSize*** value must include a header and the data size of each item to be added to the list. Use the helper macro SAI_COMPUTE_LIST_OVERHEAD(***nElement***, ***overheadSize***) to calculate the total overhead size according to the number of items the list contains. The speak list must contain at least one main header and one header per entry (speak rule).

Example

```
//
// Start synthesis with speech located in a grammar list. TTS handle is used to reach the
// appropriate synthesis engine created previously.
// The Application waits on the event queue to receive the SAIEVN_TTS_SPEAK_ACKNOWNLEDGED
and
// SAIEVN_TTS_SPEAK_DONE.
//
// A XML grammar is added to a grammar list. With saiAddSpeakRule, a name tag can be
associated
// to this grammar and used to make reference to this tool.
//
DWORD demoTtsSpeakGrammarList(  CTAQUEUEHD queueHd, SAI_SYNTHESIZER_HANDLE ttsHd )
{
    DWORD rc = SUCCESS;
    char textToSpeak[MAX_TEXT_TO_SPEAK];
    SAI_REQUEST_ID reqId = 0;
    strcpy( textToSpeak, (char *)"one one two");
    rc = saiInitSpeakList( speakRuleList, sizeof(speakRuleList) );
    if ( rc == SUCCESS )
    {
        //
        // Add a speak rule to the database.
        //
        rc = saiAddSpeakItem(speakRuleList, SAI_SPEAK_DATA_TYPE_PLAIN_TEXT, textToSpeak
);
    }
    if ( rc == SUCCESS )
    {
```

```
        //
/        / Call SAI speaking function. Use grammar list created previously.
        // reqId associated to the request is returned
        //
        rc = saiSpeakSynthesizer( ttsHd, SAI_SPEAK_DATA_TYPE_DATA_LIST, speakRuleList,
&reqId );
    }
    if ( rc == SUCCESS )
    {
        //
        // Verify each event received in queue as long as the appropriate one
        // (in this case: SAIEVN_TTS_SPEAK_ACKNOWNLEDGED and SAIEVN_TTS_SPEAK_DONE) are
        // received or before timeout is over. Verify that status code, completion cause
and
        // reqId are the same as buffer memory.
        //
        rc = demoWaitForEventSpeakFcn(queueHd, reqId);
    }
    if (rc == SUCCESS)
    {
        printf("Speak with SAI_SPEAK_DATA_TYPE_DATA_LIST succeed!.\n");
    }
    else
    {
        printf("Speak with SAI_SPEAK_DATA_TYPE_DATA_LIST failed, error code = 0x%08X.\n",
rc);
    }
    return rc;
}
```

saiNotifyBargeInToSynthesizer

Notifies the synthesizer that a barge-in occurred.

Prototype

DWORD **saiNotifyBargeInToSynthesizer** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_PROXY_SYNC_ID ***proxySyncId***, SAI_REQUEST_ID ****requestId***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.
proxySyncId	When a recognizer is used to detect a barge-in, a ***proxySyncId*** value is received from the recognizer engine through SAIEVN_START_OF_SPEECH. Use **saiNotifyBargeInToSynthesizer** to return the ***proxySyncId*** to the synthesizer.
requestId	Pointer to a unique identifier associated with the request.

Return values

Return value	Description
SUCCESS	
SAIERR_ENGINE_NOT_READY	Specified synthesizer engine is not ready to handle the request.
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.

Events

Event	Description
SAIEVN_TTS_NOTIFY_BARGE_IN_DONE	Request was processed. The result of the request is returned in the statusCode field of the SAIEVN_TTS_NOTIFY_BARGE_IN_DONE event buffer.

Details

saiNotifyBargeInToSynthesizer enables the application to inform a synthesizer when a barge-in occurrence is detected, such as detected DTMF or speech input.

Before using **saiNotifyBargeInToSynthesizer**, the application must enable the kill-on-barge-in flag with **saiTtsSetKillOnBargeIn** (set ***value*** = TRUE) prior to calling **saiSpeakSynthesizer**. Otherwise, the synthesizer rejects the call by returning a failure indication in the SAIEVN_TTS_NOTIFY_BARGE_IN_DONE event buffer.

The application must call **saiNotifyBargeInToSynthesizer** even if it receives the BARGE-IN signal from the recognizer. The application cannot assume that the signal detector and the synthesizer engine are connected. When invoking **saiNotifyBargeInToSynthesizer**, the application must specify the ***proxySyncId*** returned within SAIEVN_START_OF_SPEECH.

For more information, refer to *Implementing a barge-in enabled prompt* on page 32.

See also

saiGetResultRecognizer, **saiStartRecognizer**

saiPauseSynthesizer

Pauses a synthesizer process.

Prototype

DWORD **saiPauseSynthesizer** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_REQUEST_ID ****requestId***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.
requestId	Pointer to a unique identifier associated with the request.

Return values

Return value	Description
SUCCESS	
SAIERR_ENGINE_NOT_READY	Synthesizer associated with the specified ***synthHd*** cannot handle the request.
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.

Events

Event	Description
SAIEVN_TTS_PAUSE_DONE	Request was processed. The result of the request is returned in the **statusCode** field of the SAIEVN_TTS_PAUSE_DONE event buffer.

Details

saiPauseSynthesizer pauses a speech synthesis request that was received by the synthesizer engine.

If the synthesizer is not ready to process the request (that is, it is not in a SPEAK state), the SAI service returns SAI_SC_METHOD_NOT_VALID_IN_THIS_STATE in the statusCode field of the SAIEVN_TTS_PAUSE_DONE event buffer.

See also

saiControlSynthesizer, **saiResumeSynthesizer**, **saiSpeakSynthesizer**, **saiStopSynthesizer**

saiReleaseSynthesizer

Releases a synthesizer instance and de-allocates synthesizer engine resources.

Prototype

DWORD **saiReleaseSynthesizer** (SAI_SYNTHESIZER_HANDLE ***synthHd***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.

Return values

Return value	Description
SUCCESS	
SAIERR_ENGINE_NOT_READY	Specified synthesizer engine is not ready to handle the request.
SAIERR_INVALID_PARAMETER	***synthHd*** is invalid.

Details

saiReleaseSynthesizer de-allocates a synthesizer resource from a speech server. If **saiReleaseSynthesizer** executes successfully, the synthesizer enters a deletion state and the synthesizer object is deleted after the server responds. The SAI service returns an error in response to any subsequent requests for the specified ***synthHd***.

If the specified ***synthHd*** is not associated with a synthesizer, the handle is not freed and the SAI service returns SAIERR_INVALID_PARAMETER.

For more information, refer to *Creating synthesizer resources* on page 22.

See also

saiCreateSynthesizer

Example

```
// Destruction of TTS engine object using SAI function. Its handle is equalled to 0.
// The Application does not need to wait on the event queue to receive a specific event
// because SAI services take care to release all synthesis engine trace.
//
DWORD demoTtsRelease( SAI_SYNTHESIZER_HANDLE *ttsHd, DWORD *serverPortTts)
{
    DWORD rc = SUCCESS;

    rc = saiReleaseSynthesizer(*ttsHd );
    if ( rc == SUCCESS )
    {
       printf("Releasing TTS succeed.\n");
        *serverPortTts = 0;
        *ttsHd = 0;
    }
    else
    {
        printf("Failure on releasing TTS, error code = 0x%08x.\n", rc);
    }

    return rc;
}
```

saiResumeSynthesizer

Resumes a paused synthesizer process.

Prototype

DWORD **saiResumeSynthesizer** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_REQUEST_ID ****requestId***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.
requestId	Pointer to a unique identifier associated with the request.

Return values

Return value	Description
SUCCESS	
SAIERR_ENGINE_NOT_READY	Specified synthesizer engine is not ready to handle the request.
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.

Events

Event	Description
SAIEVN_TTS_RESUME_DONE	Request was processed. The result of the request is returned in the statusCode field of the SAIEVN_TTS_RESUME_DONE event buffer.

Details

saiResumeSynthesizer resumes a speech synthesis request that was paused with **saiPauseSynthesizer**. If the synthesizer engine is not paused when the application invokes **saiResumeSynthesizer**, the SAI service returns SAI_SC_METHOD_NOT_VALID_IN_THIS_STATE in the statusCode field of the SAIEVN_TTS_RESUME_DONE buffer.

See also

saiControlSynthesizer, **saiSpeakSynthesizer**, **saiStopSynthesizer**

saiSetParamSynthesizer

Updates the synthesizer parameters used for the duration of the current synthesizer session.

Prototype

DWORD **saiSetParamSynthesizer** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_REQUEST_ID ****requestId***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.
requestId	Pointer to a unique identifier associated with the request.

Return values

Return value	Description
SUCCESS	
SAIERR_ENGINE_NOT_READY	Specified synthesizer engine is not ready to handle the request.
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.

Events

Event	Description
SAIEVN_SET_PARAMETER_DONE	Request was processed. The result of the request is returned in the statusCode field of the SAIEVN_SET_PARAMETER_DONE event buffer.

Details

If required, the application must set specific speech parameters with synthesizer parameter functions such as **saiTtsSetKillOnBargeIn** and **saiTtsSetSpeakerProfile** prior to invoking **saiSetParamSynthesizer**. For more information, refer to *Managing USAI parameters* on page 29.

See also

saiGetParamSynthesizer

saiSpeakSynthesizer

Initiates a speech synthesis process.

Prototype

DWORD **saiSpeakSynthesizer** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_SPEAK_DATA_TYPE ***dataType***, void ****data***, SAI_REQUEST_ID ****requestId***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.
dataType	Payload data type. The following list shows supported speak data types and their associated numeric values: • SAI_SPEAK_DATA_TYPE_URL [0]: Data in URL format. • SAI_SPEAK_DATA_TYPE_XML [1]: Data in XML format. • SAI_SPEAK_DATA_TYPE_PLAIN_TEXT [2]: Data in non-formatted plain text format. • SAI_SPEAK_DATA_TYPE_DATA_LIST [3]: Data is a list of speak rules.
data	Pointer to the payload to play. This argument can consist of a URL, an XML document, a created speak grammar list, or plain text.
requestId	Pointer to a unique identifier associated with the request.

Return values

Return value	Description
SUCCESS	
SAIERR_ENGINE_NOT_READY	Specified synthesizer engine is not ready to handle the request.
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_LIST_EMPTY	Speak list contains no item.
SAIERR_LIST_NOT_INITIALIZED	Specified speak list is not initialized. This error is only returned when ***datatype*** is SAI_SPEAK_DATA_TYPE_DATA_LIST.

Events

Event	Description
SAIEVN_TTS_SPEAK_ACKNOWLEDGED	Request was accepted and is being processed. Status code is indicated in the statusCode field of the SAIEVN_TTS_SPEAK_ACKNOWLEDGED event buffer.
SAIEVN_TTS_SPEAK_DONE	If the speech server successfully processes the command, the application receives this event indicating that the request is completed. The completion cause is returned in the completionCause field of the SAIEVN_TTS_SPEAK_DONE event buffer.
SAIEVN_TTS_SPEAK_MARKER_REACHED	Unsolicited event that may be returned by the synthesizer when markers are included into the speech data. The marker is returned in the SAIEVN_TTS_SPEAK_MARKER_REACHED event buffer.

Details

If the synthesizer is actively processing data or is paused when it receives the speech synthesis request, the synthesizer accepts the request. The SAI service returns an event indicating that the request status is PENDING. Otherwise, the event indicates the request status is IN_PROGRESS.

For more information, refer to *Creating a synthesizer speak list* on page 26.

See also

saiControlSynthesizer, **saiPauseSynthesizer**, **saiResumeSynthesizer**, **saiStopSynthesizer**

Example

```
//
// Start synthesis with speech located in a grammar list. TTS handle is used to reach the
// appropriate synthesis engine created previously.
// The Application waits on the event queue to receive the SAIEVN_TTS_SPEAK_ACKNOWNLEDGED
and
// SAIEVN_TTS_SPEAK_DONE.
//
// A XML grammar is added to a grammar list. With saiAddSpeakRule, a name tag can be
associated
// to this grammar and used to make reference to this tool.
//
DWORD demoTtsSpeakGrammarList(  CTAQUEUEHD queueHd, SAI_SYNTHESIZER_HANDLE ttsHd )
{
    DWORD rc = SUCCESS;
    char textToSpeak[MAX_TEXT_TO_SPEAK];
    SAI_REQUEST_ID reqId = 0;
    strcpy( textToSpeak, (char *)"one one two");
    rc = saiInitSpeakList( speakRuleList, sizeof(speakRuleList) );
    if ( rc == SUCCESS )
    {
        //
        // Add a speak rule to the database.
        //
        rc = saiAddSpeakItem(speakRuleList, SAI_SPEAK_DATA_TYPE_PLAIN_TEXT, textToSpeak
);
    }
    if ( rc == SUCCESS )
    {
        //
        // Call SAI speaking function. Use grammar list created previously.
        // reqId associated to the request is returned
        //
        rc = saiSpeakSynthesizer( ttsHd, SAI_SPEAK_DATA_TYPE_DATA_LIST, speakRuleList,
&reqId );
    }
    if ( rc == SUCCESS )
    {
        //
        // Verify each event received in queue as long as the appropriate one
        // (in this case: SAIEVN_TTS_SPEAK_ACKNOWNLEDGED and SAIEVN_TTS_SPEAK_DONE) are
        // received or before timeout is over. Verify that status code, completion cause
and
        // reqId are the same as buffer memory.
        //
        rc = demoWaitForEventSpeakFcn(queueHd, reqId);
    }
    if (rc == SUCCESS)
    {
        printf("Speak with SAI_SPEAK_DATA_TYPE_DATA_LIST succeed!.\n");
    }
    else
    {
        printf("Speak with SAI_SPEAK_DATA_TYPE_DATA_LIST failed, error code = 0x%08X.\n",
```

```
rc);
    }

    return rc;
}
```

saiStopSynthesizer

Stops a speech synthesis process.

Prototype

DWORD **saiStopSynthesizer** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_REQUEST_ID ****requestIdList***, SAI_REQUEST_ID ****requestId***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.
requestIdList	Pointer to a table of request IDs to be stopped. The last element must be NULL. If ***requestIdList*** is set to NULL, all speak commands queued will be stopped.
requestId	Pointer to a unique identifier associated with the request.

Return values

Return value	Description
SUCCESS	
SAIERR_ENGINE_NOT_READY	Specified synthesizer engine is not ready to handle the request.
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.

Events

Event	Description
SAIEVN_TTS_STOP_DONE	Request was processed. The result of the request is returned in the statusCode field of the SAIEVN_TTS_STOP_DONE event buffer.

Details

saiStopSynthesizer stops one or more synthesizer requests associated with a synthesizer engine. Applications use ***requestIdList*** to specify a list of synthesizer requests to stop. The SAI service provides a list of the stopped request IDs in the reqList field of the SAIEVN_TTS_STOP_DONE event buffer.

See also

saiControlSynthesizer, **saiPauseSynthesizer**, **saiResumeSynthesizer**, **saiSpeakSynthesizer**

Example

```
//
// Stop all active synthesizing process. TTS handle is used to reach the appropriate
// synthesis engine created and started previously.
//
// This function can be modified to stop only specified speech command (IN_PROGRESS or
PENDING).
// So, a reqId list containing reqId returned after a start recognizing command must be
passed
// as second argument when calling saiStopSynthesizer.
//
DWORD demoTtsStop( CTAQUEUEHD queueHd, SAI_SYNTHESIZER_HANDLE ttsHd)
{
    DWORD rc = SUCCESS;
    int i = 0;
    CTA_EVENT event;
```

```
    SAI_REQUEST_ID reqId = 0;
    SAI_EVENT_TTS_STOP* eventResult = NULL;

    memset( &event, 0, sizeof (CTA_EVENT) );
    //
    // This procedure will stop all requests. reqId associated to current request is
returned.
    //
    rc = saiStopSynthesizer( ttsHd, NULL, &reqId );
    if( rc == SUCCESS )
    {
//
        // Verify each event received in queue as long as the appropriate one
        // (in this case: SAIEVN_TTS_STOP_DONE) is received or before timeout
        // is over.
        //
        rc = demoWaitForEvent(queueHd, &event, SAIEVN_TTS_STOP_DONE, 0 );
    }
    if ( rc == SUCCESS )
    {
        //
        // The buffer pointer contains a speech access event. Cast buffer according to
the
        // event type.
        //
        eventResult = (SAI_EVENT_TTS_STOP*)(event.buffer) ;
        //
        // Verify that reqId and statusCode are the same as buffer memory.
        //
        if( (eventResult != NULL) &&
            (reqId == eventResult->header.reqId)  &&
            (eventResult->statusCode == SAI_SC_SUCCESS) )
        {
            //
            // Release the allocated internal memory
            //
            if( (event.buffer != NULL) && (event.size & CTA_INTERNAL_BUFFER) )
            {
                dispFreeBuffer ( event.buffer );
            }
        }
        else
        {
            printf("Status Code = 0x%08x.\n", eventResult->statusCode);
            rc = !SUCCESS;
        }
    }
    if (rc == SUCCESS)
    {
        printf("Stop succeed. Here is the stoped request ID list.\n");
        for (i=0; i != 0; i++)
        {
             printf("%d\n", eventResult->reqList[i]);
        }
    }
    else
    {
        printf("Stop failed, error code = 0x%08x.\n");
    }
    return rc;
}
```

10 Synthesizer parameter function reference

Using the synthesizer parameter function reference

This section provides an alphabetical reference to the synthesizer parameter functions in the Universal Speech Access API . A typical function includes:

Prototype	The prototype is followed by a list of the function arguments. Data types include: • WORD (16-bit unsigned) • DWORD (32-bit unsigned) • INT16 (16-bit signed) • INT32 (32-bit signed) • BYTE (8-bit unsigned) If a function argument is a data structure, the complete data structure is defined. Refer to *Universal Speech Access API structures* on page 163 for more information.
Return values	The return value for a function is either SUCCESS or an error code. For asynchronous functions, a return value of SUCCESS indicates the function was initiated; subsequent events indicate the status of the operation. Refer to *Universal Speech Access API errors* on page 139 for more information.
Events	If events are listed, the function is asynchronous and is complete when the DONE event is returned. If there are no events listed, the function is synchronous. Additional information such as reason codes and return values may be provided in the value field of the event. Each event consists of an event ID, an event header value field, and the informational event buffer. Refer to *Universal Speech Access API events* on page 141 for more information.
Example	Example functions taken from sample application programs shipped with the product. The notation `/* ... */` indicates additional code that is not shown.

saiTtsAddVendorSpecific

Adds vendor-specific synthesizer parameters to the local database.

Prototype

DWORD **saiTtsAddVendorSpecific** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_VENDOR_SPECIFIC ***vp***[], DWORD ***count***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.
vp	Array of vendor-specific parameters to add to the local database. Only the vp[x].vendorSpecificName parameter must be specified.
count	Number of vendor-specific parameters to add. This value must not exceed the size of the ***vp***[]).

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.

Details

Before retrieving or setting vendor-specific parameters, the application must first add them to the local database with **saiTtsAddVendorSpecific**. These settings are not transferred to the speech server until the application calls **saiSetParamSynthesizer**.

See also

saiTtsGetVendorSpecific, **saiTtsSetVendorSpecific**

saiTtsGetAudioFetchHint

Retrieves whether or not the synthesizer can optimize speech by pre-fetching audio.

Prototype

DWORD **saiTtsGetAudioFetchHint** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_AUDIO_FETCH_HINT ****hint***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.
hint	Pointer to a value that indicates whether the synthesizer can attempt to optimize speech by pre-fetching audio. Allowed values are: • SAI_AUDIO_FETCH_HINT_PREFETCH [0]: Permit, but not require the synthesizer to pre-fetch audio. • SAI_AUDIO_FETCH_HINT_SAFE [1]: Pre-fetch audio only when needed (not before). • SAI_AUDIO_FETCH_HINT_STREAM [2]: Enable the synthesizer to stream any audio fetches.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Speech engine does not support the specified parameter.

See also

saiTtsSetAudioFetchHint

saiTtsGetFetchHint

Retrieves when the synthesizer retrieves content from a document server.

Prototype

DWORD **saiTtsGetFetchHint** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_FETCH_HINT ****hint***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.
hint	Pointer to a value that indicates when the synthesizer can retrieve documents from another resource. Allowed values are: • SAI_FETCH_HINT_PREFETCH [0]: A file can be downloaded when the request is received. • SAI_FETCH_HINT_SAFE [1]: Files are only downloaded when required.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Speech engine does not support the specified parameter.

Details

Use **saiTtsGetFetchHint** to set the URL access properties to apply when the synthesizer needs to fetch documents or other resources such as speech markup or audio files.

See also

saiTtsSetFetchHint

saiTtsGetFetchTimeout

Retrieves the synthesizer timeout period for waiting for resources from the network.

Prototype

DWORD **saiTtsGetFetchTimeout** (SAI_SYNTHESIZER_HANDLE ***synthHd***, DWORD ****timeout***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.
timeout	Pointer to a timeout value in ms.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Speech engine does not support the specified parameter.

See also

saiTtsSetFetchTimeout

saiTtsGetKillOnBargeIn

Retrieves whether or not kill-on-barge-in support is enabled for a specified synthesizer.

Prototype

DWORD **saiTtsGetKillOnBargeIn** (SAI_SYNTHESIZER_HANDLE ***synthHd***, BOOL ****value***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.
value	Pointer to one of the following values: • TRUE: Kill-on-barge-in support is enabled. • FALSE: Kill-on-barge-in support is disabled.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Speech engine does not support the specified parameter.

Details

When kill-on-barge-in is enabled, the synthesizer stops all active or pending speech requests when it is notified of a barge-in condition. If the recognizer and the synthesizer are in the same session, the recognizer engine can notify the synthesizer engine of barge-in detection and ask the synthesizer to immediately stop all speech requests. In both cases, the application performs the same action, except that the synthesizer is notified earlier when the recognizer and synthesizer are in the same session.

For more information, refer to *Implementing a barge-in enabled prompt* on page 32.

See also

saiTtsSetKillOnBargeIn

saiTtsGetProsody

Retrieves synthesizer speech prosody parameters such as pitch, duration, and volume.

Prototype

DWORD **saiTtsGetProsody** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_PROSODY ****value***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.

value	Pointer to the SAI_PROSODY structure that indicates speaker voice prosody characteristics such as pitch, duration, and volume:
	`typedef struct`
	`{`
	`  INT32  pitch;`
	`  INT32  contour;`
	`  INT32  range;`
	`  INT32  rate;`
	`  INT32  duration;`
	`  INT32  volume;`
	`}`
	`SAI_PROSODY;`
	For more information, refer to *SAI_PROSODY* on page 163. For information about prosody attributes, refer to the SSML specification and to the vendor documentation.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Speech engine does not support the specified parameter.

See also

saiTtsSetProsody

saiTtsGetSpeakerProfile

Retrieves the URL where the speaker profile is kept.

Prototype

DWORD **saiTtsGetSpeakerProfile** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_URL ***url***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.
url	URL that indicates where the speaker profile is located.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Speech engine does not support the specified parameter.

Details

The speaker profile consists of a set of voice parameters such as gender and accent. For information about the speaker profile format, refer to the vendor documentation.

See also

saiTtsSetSpeakerProfile

saiTtsGetSpeechLanguage

Retrieves the default language used when synthesizing speech.

Prototype

DWORD **saiTtsGetSpeechLanguage** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_SPEECH_LANGUAGE ****language***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.

language	Pointer to the SAI_SPEECH_LANGUAGE structure:
	`typedef struct`
	`{`
	`  SAI_SPEECH_LANGUAGE  language;`
	`  SAI_SPEECH_LANGUAGE_STRING  languageString;`
	`}`
	`SAI_SPEECH_LANGUAGE;`
	For more information, refer to *SAI_SPEECH_LANGUAGE* on page 164.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Speech engine does not support the specified parameter.

See also

saiTtsSetSpeechLanguage

saiTtsGetVendorSpecific

Retrieves vendor-specific synthesizer parameters from the local database.

Prototype

DWORD **saiTtsGetVendorSpecific** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_VENDOR_SPECIFIC ***vp***[], DWORD ***count***, SAI_VENDOR_SPECIFIC_OUT ***vpOut***[])

Argument	Description
synthHd	Handle associated with a synthesizer resource.
vp	Array of vendor-specific parameters to retrieve from the local database. Only the vp[***x***].vendorSpecificName parameter must be specified.
count	Number of vendor-specific parameters to retrieve.
vpOut	Array of pointers that specify where to store retrieved vendor-specific parameters.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_VENDOR_SPECIFIC_ERROR	At least one of the specified parameters was not previously added to the local database.

Details

Before retrieving vendor-specific parameters, the application must first add them to the local database with **saiTtsAddVendorSpecific** and retrieve the specified values from the server with **saiGetParamSynthesizer**.

Note: The ***vp***[] and ***vpOut***[] arrays must be the same size and the number of specified parameters must not exceed the size of ***vp***[] and ***vpOut***[].

See also

saiTtsSetVendorSpecific

saiTtsGetVoice

Retrieves speaker voice characteristics based on Speech Synthesis Markup Language (SSML) specifications.

Prototype

DWORD **saiTtsGetVoice** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_VOICE ****value***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.

value	Pointer to the SAI_VOICE structure that specifies the speaker voice attribute:
	`typedef struct`
	`{`
	`  SAI_VOICE_GENDER  gender;`
	`  INT32  age;`
	`  char  variant[SAI_MAX_STRINGLENGTH];`
	`  char  name[SAI_MAX_STRINGLENGTH];`
	`}`
	`SAI_VOICE;`
	For more information, refer to *SAI_VOICE* on page 165. For information about speaker attributes, refer to the SSML specification and to the vendor documentation.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Speech engine does not support the specified parameter.

See also

saiTtsSetVoice

saiTtsSetAudioFetchHint

Specifies whether or not the synthesizer can optimize speech by pre-fetching audio.

Prototype

DWORD **saiTtsSetAudioFetchHint** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_AUDIO_FETCH_HINT ***hint***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.
hint	Value that specifies if the synthesizer can attempt to optimize speech by pre-fetching audio. Allowed values are: • SAI_AUDIO_FETCH_HINT_PREFETCH [0]: Permit, but not require the synthesizer to pre-fetch audio. • SAI_AUDIO_FETCH_HINT_SAFE [1]: Pre-fetch audio only when needed (not before). • SAI_AUDIO_FETCH_HINT_STREAM [2]: Enable the synthesizer to stream any audio fetches.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Speech engine does not support the specified parameter.

See also

saiTtsGetAudioFetchHint

saiTtsSetFetchHint

Specifies when the synthesizer retrieves content from a document server.

Prototype

DWORD **saiTtsSetFetchHint** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_FETCH_HINT ***hint***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.
hint	Value that specifies when the synthesizer can retrieve documents from another resource. Allowed values are: • SAI_FETCH_HINT_PREFETCH [0]: A file can be downloaded when the request is received. • SAI_FETCH_HINT_SAFE [1]: Files are downloaded only when required.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Speech engine does not support the specified parameter.

See also

saiTtsGetFetchHint

saiTtsSetFetchTimeout

Specifies the synthesizer timeout period for waiting for resources from the network.

Prototype

DWORD **saiTtsSetFetchTimeout** (SAI_SYNTHESIZER_HANDLE ***synthHd***, DWORD ***timeout***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.
timeout	Timeout value in ms.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Speech engine does not support the specified parameter.

See also

saiTtsGetFetchTimeout

saiTtsSetKillOnBargeIn

Enables or disables kill-on-barge-in support for a specified synthesizer.

Prototype

DWORD **saiTtsSetKillOnBargeIn** (SAI_SYNTHESIZER_HANDLE ***synthHd***, BOOL ***value***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.
value	One of the following values: • TRUE: Kill-on-barge-in support is enabled. • FALSE: Kill-on-barge-in support is disabled.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Speech engine does not support the specified parameter.

Details

When kill-on-barge-in is enabled, the synthesizer stops all active or pending speech requests when it is notified of a barge-in condition. If the recognizer and the synthesizer are in the same session, the recognizer engine can notify the synthesizer engine of barge-in detection and ask the synthesizer to immediately stop all speech requests. In both cases, the application performs the same action, except that the synthesizer is notified earlier when the recognizer and synthesizer are in the same session.

Note: In either case, SAIEVN_START_OF_SPEECH is sent to the application. The application must close the loop by calling **saiNotifyBargeInToSynthesizer** and specifying the ***proxySyncId*** received in SAIEVN_START_OF_SPEECH.

For more information, refer to *Implementing a barge-in enabled prompt* on page 32.

See also

saiTtsGetKillOnBargeIn

saiTtsSetProsody

Specifies synthesizer speech prosody parameters such as pitch, duration, and volume.

Prototype

DWORD **saiTtsSetProsody** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_PROSODY ***value***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.

value	SAI_PROSODY structure that defines speaker voice prosody characteristics such as pitch, duration, and volume:
	`typedef struct`
	`{`
	`  INT32  pitch;`
	`  INT32  contour;`
	`  INT32  range;`
	`  INT32  rate;`
	`  INT32  duration;`
	`  INT32  volume;`
	`}`
	`SAI_PROSODY;`
	For more information, refer to *SAI_PROSODY* on page 163. For information about prosody attributes, refer to the SSML specification and to the vendor documentation.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Speech engine does not support the specified parameter.

See also

saiTtsGetProsody

saiTtsSetSpeakerProfile

Sets the URL where the profile of the speaker is kept.

Prototype

DWORD **saiTtsSetSpeakerProfile** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_URL ***url***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.
url	URL that indicates where the speaker profile is located.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Speech engine does not support the specified parameter.

Details

The speaker profile consists of a set of voice parameters such as gender and accent. For information about the speaker profile format, refer to the vendor documentation.

See also

saiTtsGetSpeakerProfile

saiTtsSetSpeechLanguage

Specifies the default language used when synthesizing speech.

Prototype

DWORD **saiTtsSetSpeechLanguage** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_SPEECH_LANGUAGE ***language***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.

language	SAI_SPEECH_LANGUAGE structure:
	`typedef struct`
	`{`
	`  SAI_SPEECH_LANGUAGE  language;`
	`  SAI_SPEECH_LANGUAGE_STRING  languageString;`
	`}`
	`SAI_SPEECH_LANGUAGE;`
	For more information, refer to *SAI_SPEECH_LANGUAGE* on page 164.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Speech engine does not support the specified parameter.

See also

saiTtsGetSpeechLanguage

saiTtsSetVendorSpecific

Specifies vendor-specific synthesizer parameters for the local database.

Prototype

DWORD **saiTtsSetVendorSpecific** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_VENDOR_SPECIFIC ***vp***[], DWORD ***count***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.
vp	Array of vendor-specific parameter values to set in the local database.
count	Number of vendor specific parameters to specify. This value indicates the number of entries in ***vp***[].

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_VENDOR_SPECIFIC_ERROR	At least one of the provided parameters was not previously added to the local database.

Details

Before setting vendor-specific parameters, the application must first add them to the local database with **saiTtsAddVendorSpecific** and retrieve the specified values from the server with **saiSetParamSynthesizer**.

See also

saiTtsGetVendorSpecific

saiTtsSetVoice

Specifies speaker voice characteristics based on SSML specifications. For more information, refer to *Managing USAI parameters* on page 29.

Prototype

DWORD **saiTtsSetVoice** (SAI_SYNTHESIZER_HANDLE ***synthHd***, SAI_VOICE ***value***)

Argument	Description
synthHd	Handle associated with a synthesizer resource.

value	Pointer to the SAI_VOICE structure that specifies the speaker voice attribute:
	`typedef struct`
	`{`
	`  SAI_VOICE_GENDER  gender;`
	`  INT32  age;`
	`  char  variant[SAI_MAX_STRINGLENGTH];`
	`  char  name[SAI_MAX_STRINGLENGTH];`
	`}`
	`SAI_VOICE;`
	For more information, refer to *SAI_VOICE* on page 165. For information about speaker attributes, refer to the SSML specification and to the vendor documentation.

Return values

Return value	Description
SUCCESS	
SAIERR_INVALID_PARAMETER	One of the specified arguments is invalid.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Speech engine does not support the specified parameter.

See also

saiTtsGetVoice

11 Demonstration program

Using saidemo

saidemo is a sample application that enables you to create recognizer (ASR) and synthesizer (TTS) resources located on remote speech servers. Once created, *saidemo* provides a set of options to control the resources.

Purpose

Demonstrates the speech interface provided by the SAI service. The application initializes the Natural Access environment, opens the appropriate services (ADI, MSPP, SAI), and enables you to choose between two sets of options:

	Description
Unit test	Executes individual SAI service functions.
Integration test	Executes a series of function calls that perform a complete set of test processes.

The unit test and the integration test enable you to:

- Create, control, and destroy recognizer and synthesizer resources.
- View and modify synthesizer and recognizer engine parameters.

Usage

```
saidemo arguments
```

Featured functions

saiAsrAddVendorSpecific, **saiAsrGetVendorSpecific**, **saiAsrSetVendorSpecific**, **saiCreateRecognizer**, **saiCreateSynthesizer**, **saiGetParamRecognizer**, **saiGetParamSynthesizer**, **saiReleaseRecognizer**, **saiReleaseSynthesizer**, **saiSpeakSynthesizer**, **saiStartRecognizer**, **saiStopRecognizer**, **saiStopSynthesizer**, **saiTtsAddVendorSpecific**, **saiTtsGetVendorSpecific**, **saiTtsSetVendorSpecific**

Prerequisites

Before running *saidemo*, modify the following files as necessary:

Configuration file	Sample file	Description
NaturalAccess configuration file	*cta.cfg*	Specifies that Natural Access initialize the SAI service and the SAI service manager when the system boots. In addition, other services used in the application must be initialized in this file (for example, the ADI and MSPP services). Location: • Windows: c:\NMS\CTAccess\CFG\cta.cfg • UNIX: /opt/nms/ctaccess/cfg/cta.cfg
SAI configuration file	*saiconfig.xml*	Specifies parameters that define which speech servers to access on the network. Location: • Windows: c:\NMS\CTAccess\CFG\saiconfig.xml • UNIX: /opt/nms/ctaccess/cfg/saiconfig.xml
CG board keyword file	*cg6kfusion-t.cfg*	Specifies board-level parameters that determine how board resources are allocated. Location: • Windows: c:\NMS\CG\Cfg\cg6kxxx.cfg • UNIX: /opt/nms/cg/cfg/cg6kxxx.cfg
OAM system configuration file	*oamsys.cfg*	Specifies bus and slot numbers for each board to match the chassis configuration. Location: • Windows: c:\NMS\OAM\cfg\oamsys.cfg • UNIX: /opt/nms/oam/cfg/oamsys.cfg

Procedure

To run *saidemo*:

Step	Action
1	Modify or create the appropriate configuration files according to the system configuration.
2	Open a command line window and navigate to the *nms**spaccess**samples**saidemo* directory.
3	Enter the following command:

Step	Action
	`saidemo -s`***`server`*** `-i`***`SRV_IP`*** `-b`***`CG_IP_1`*** `-t`***`TTS_RTP_PRT`*** `-a`***`ASR_RTP_PRT`** where: ** **Argument** | **Description** -s ***server*** | Speech server name defined in the *saiconfig.xml* file. -i ***SRV_IP*** | IP address of a speech server. -b ***CG_IP_1*** | IP address of the CG board. -t ***TTS_RTP_PRT*** | RTP port to use for synthesizer process. -a ***ASR_RTP_PRT*** | RTP port to use for recognizer process. -h | View a list of program options. For example: `saidemo -sMRCP1 -i10.10.20.54 -b10.10.35.112 -t2000 -a2002`

Details

The following table shows *saidemo* initialization steps:

Step	Description
1	Initialize Natural Access environment.
2	Create a Natural Access event queue and return its handle.
3	Create a context for the current queue and return its handle.
4	Initialize the SAI, ADI, and MSPP services and open them in the current context.

When you run *saidemo*, it displays the following menu options:

Option	Description
0. Exit this application	Closes all resources, closes all services, destroys context and queue, and exits the program.
1. Unit Tests	Sub-menu that provides options for executing individual SAI service functions.
2. Integration Tests	Sub-menu that provides options for performing complete sets of test processes.

Unit tests

Unit test provides the following options:

Option	Description
0. Exit this menu	Exits the unit test menu and returns to the main menu. This option frees all resources that have not been released.
1. Print context	Prints the state of the message queue, the Natural Access context, and any recognizer or synthesizer handle. Indicates if Natural Access services have been opened.
2. Create tts	Creates a synthesizer resource in the current Natural Access context according to the parameters specified at the command line.
3. Get tts parameters	Lists all synthesizer (TTS) parameters.

Option	Description
4. Set tts parameters	Opens a submenu that displays options for individually setting synthesizer parameters. Before exiting the submenu, specify whether to transfer the new synthesizer parameters to the speech server (for continued use during the current synthesizer session) or apply the modified parameters only to the next command. For information about applying modified parameters, refer to *Managing USAI parameters* on page 29.
5. Start tts (SPEAK)	Sends a synthesizer process request to the speech server after displaying the following speech options: • Synthesize speech from a sentence from the keyboard input. • Synthesize speech from text at a specific URL. • Synthesize speech from a pre-defined grammar list. • Synthesize speech from an embedded server grammar. When the command completes, *saidemo* displays the status of the synthesizer process.
6. Stop tts	Sends a request to the speech server to stop the current synthesizer process.
7. Release tts	Releases the synthesizer resource and resets the synthesizer handle.
8. Create asr	Creates a recognizer in the current Natural Access context according to the parameters specified at the command line.
9. Start asr	Sends a recognize command to the speech server after displaying the following grammar type options: • Perform speech recognition from speech data located at a specific URL. • Perform speech recognition from speech data located at a list of URLs. • Perform speech recognition from speech data from a pre-defined grammar list. • Perform speech recognition from speech data from a grammar list name. • Perform speech recognition from speech data from an embedded server grammar. • Perform speech recognition from multiple-part define grammar. • Perform speech recognition from multiple-part recognition. When the command completes, *saidemo* displays the status of the speech recognition process.
10. Get asr parameters	List all recognition (ASR) parameters.
11. Set asr parameters	Opens a submenu that displays options for individually setting recognizer parameters. Before exiting the submenu, specify whether to transfer the new recognizer parameters to the speech server (for continued use during the current recognizer session) or to apply the modified parameters only to the next command. For information about applying modified parameters, refer to *Managing USAI parameters* on page 29.
12. Stop asr	Sends a request to the speech server to stop the current speech recognition process.
13. Release asr	Releases the synthesizer resource and resets the recognizer handle.

Integration tests

Integration test provides the following options:

Option	Description
0. Exit this menu	Exits the integration test menu and returns to the main menu. This option frees all resources that have not been released.
1. Create a TTS and its RTP endpoint	Creates a synthesizer and its media connection (Fusion RTP endpoint).
2. Destroy tts and its RTP endpoint	Destroys the previously created synthesizer and its media connection.
3. Create asr and its RTP endpoint	Creates a recognizer and its media connection (the Fusion RTP endpoint).
4. Destroy asr and its RTP endpoint	Destroys the previously created recognizer and its media connection.
5. Cross connection using t1_loop_back configuration type	Cross-connects a synthesizer with a recognizer engine using the t1_loop_back configuration.
6. Destroy t1_loop_back connection	Destroys all resources and resets any handles created by option 5.
7. Cross connection using one_board_c_over configuration type	Cross-connects a synthesizer with a recognizer engine using the one_board_c_over configuration.
8. Destroy one_board_c_over connection	Destroys all resources and resets any handles created by option 7.
9. Speak and recognize	Once the cross-connection is created (option 5 or 7) a RECOGNIZE command is sent to the recognizer (ASR). When the RECOGNIZE acknowledge event is received, a SPEAK command is sent to the synthesizer (TTS) to start synthesizing speech. The application waits for a recognition complete event and displays the results.

12 Errors, events, and parameters

Universal Speech Access API errors

This topic provides a summary of Universal Speech Access API error codes. All USAI functions return a status code. The return code is either SUCCESS (0) or an error code indicating that the function failed and a reason for the failure.

Universal Speech Access API error codes are defined in the *saidef.h and ctaerr.h* include files. The error codes are prefixed with SAIERR_ or CTAERR_.

Alphabetical listing

The following table alphabetically lists the USAI errors (all errors are 32 bits):

Error name	Description
SAIERR_ENGINE_FAILED	Failed to create a new USAI speech engine object.
SAIERR_ENGINE_NOT_READY	Application attempted to use a synthesizer engine that is not in a ready state.
SAIERR_FAILED	General error code failure occurred.
SAIERR_FAILED_TO_GET_LICENCE	License file was not present when the application attempted to open the USAI with **ctaInitialize**.
SAIERR_FAILED_TO_QUEUE_EVENT	USAI failed to queue an event in the NaturalAccess queue.
SAIERR_INVALID_PARAMETER	Value specified for one of the mandatory arguments is invalid.
SAIERR_LIST_EMPTY	Specified grammar list contains no entries.
SAIERR_LIST_NOT_INITIALIZED	Application attempted to use a non-initialized grammar list.
SAIERR_NO_MORE_LICENSED_PORT_AVAILABLE	Application exceeded the maximum number of licensed ports when invoking **saiCreateSynthesizer** or **saiCreateRecognizer**.
SAIERR_NOT_ENOUGH_MEMORY	Not enough memory space when attempting to initialize or format speech grammar lists.
SAIERR_UNSUPPORTED_SPEECH_PARAMETER	Application specified an unsupported vendor parameter.
SAIERR_VENDOR_SPECIFIC_ERROR	At least one of the parameters specified in a vendor-specific parameter function was not previously added to the local database.

Numerical listing

The following table numerically lists the USAI errors:

Hex	Decimal	Error name
0x320001	3276801	SAIERR_ENGINE_FAILED
0x320002	3276802	SAIERR_ENGINE_NOT_READY
0x320003	3276803	SAIERR_INVALID_PARAMETER
0x320004	3276804	SAIERR_LIST_NOT_INITIALIZED
0x320005	3276805	SAIERR_NOT_ENOUGH_MEMORY
0x320006	3276806	SAIERR_FAILED
0x320007	3276807	SAIERR_UNSUPPORTED_SPEECH_PARAMETER
0x320008	3276808	SAIERR_FAILED_TO_QUEUE_EVENT
0x320009	3276809	SAIERR_VENDOR_SPECIFIC_ERROR
0x32000A	3276810	SAIERR_NO_MORE_LICENSED_PORT_AVAILABLE
0x32000B	3276811	SAIERR_FAILED_TO_GET_LICENCE
0x32000D	3276813	SAIERR_LIST_EMPTY

Universal Speech Access API events

In response to commands executed by an application, USAI generates events and reason codes that indicate the occurrence (or lack) of certain conditions or state changes.

This topic provides the following information:

- NaturalAccess event structure
- Universal Speech Access API event header
- Alphabetical event summary
- Numerical event summary
- Event information buffers

NaturalAccess event structure

Events returned by NaturalAccess APIs (such as USAI) conform to the following structure:

```
typedef struct
{
DWORD     id;
CTAHD     ctahd;
DWORD     timestamp;
DWORD     userid;
DWORD     size;
void      *buffer;
DWORD     value;
unsigned  objHd;
} CTA_EVENT;
```

where fields in the event structure provide the following information:

Field	Description
id	Event code (and source service id).
ctahd	NaturalAccess context handle.
timestamp	Timestamp associated with the event.
userid	User id defined by a call to **ctaCreateContext**.
size	Size of buffer if buffer, if applicable.
buffer	Pointer to a buffer location containing additional information associated with a the event
value	Event status information or event-specific data.
objHd	Service object handle.

USAI events are preceded with the prefix SAIEVN_ and contain. All defined SAI event buffers contain event header information and may contain additional information in the event buffer.

Universal Speech Access API event header

Universal Speech Access API events contain the following header:

```
typedef struct
{
DWORD              size;         /* For compatibility. */
DWORD              eventType;    /* Event type */
SAI_REQUEST_ID     reqId;        /* Request id associated to this event */
SAI_REQUEST_STATE  reqState;     /* Command request state */
DWORD              engineHandle; /* SAI_SYNTHESIZER_HANDLE or SAI_RECOGNIZER_HANDLE */

DWORD              clientObjId;  /* The client registered object identifier */
}
SAI_EVENT_HEADER;
```

The USAI event header provides the following information:

Field	Description
size	Size of the structure.
eventType	Type of event.
reqId	Request ID (***requestId***) associated with the event.
reqState	State of the command request associated with the event. Valid states: • In-progress • Pending • Complete
engineHandle	Type of resource engine to which the request was directed. Valid engine types: • SAI_SYNTHESIZER_HANDLE • SAI_RECOGNIZER_HANDLE
clientObjId	Client registered object identifier.

Alphabetical listing

The following table alphabetically lists the USAI events:

Event name	Description
SAIEVN_ASR_DEFINE_GRAMMAR_DONE	**saiDefineGrammarRecognizer** request was processed. The result of the request is returned in the statusCode field of the event buffer. Returns additional information in a SAI_EVENT_ASR_DEFINE_GRAMMAR buffer.
SAIEVN_ASR_GET_RESULT_DONE	**saiGetResultRecognizer** request was processed. The result of the request is returned in the statusCode field of the event buffer. Returns additional information in a SAI_EVENT_ASR_GET_RECOGNITION_RESULT buffer.
SAIEVN_ASR_RECOGNITION_DONE	**saiStartRecognizer** request is complete. The completion cause is provided in the completionCause field of the event buffer. Returns additional information in a SAI_EVENT_ASR_RECOGNITION_COMPLETE buffer.
SAIEVN_ASR_RECOGNITION_STARTED	Request to start the recognizer with **saiStartRecognizer** is currently in process. Returns additional information in a SAI_EVENT_COMMON buffer.

Event name	Description
SAIEVN_ASR_START_OF_SPEECH	Recognizer detected speech input. Returns additional information in a SAI_EVENT_ASR_START_OF_SPEECH buffer.
SAIEVN_ASR_START_TIMER_DONE	**saiStartTimerRecognizer** request was processed. The result of the request is returned in the statusCode field of the event buffer. Returns additional information in a SAI_EVENT_COMMON buffer.
SAIEVN_ASR_STOP_DONE	**saiStopRecognizer** request was processed. The result of the request is returned in the statusCode field of the event buffer. Returns additional information in a SAI_EVENT_ASR_STOP buffer.
SAIEVN_ENGINE_CREATION_DONE	**saiCreateRecognizer** or **saiCreateSynthesizer** completed. The success or failure of the operation is indicated in the statusCode field of the event buffer. If the command completed successfully (the status code is SAI_SC_SUCCESS), the application can use the returned handle (***recogHd*** or ***synthHd***). If the operation failed, the application must call **saiReleaseRecognizer** or **saiReleaseSynthesizer** to release the returned handle. Returns additional information in a SAI_EVENT_ENGINE_CREATION buffer.
SAIEVN_ENGINE_RELEASE_DONE	USAI released the specified recognizer or synthesizer engine. Returns additional information in a SAI_EVENT_ENGINE_RELEASE buffer.
SAIEVN_ENGINE_FAILED	The connection to a TTS or ASR engine has failed. returns additional information in a SAI_EVENT_ENGINE_FAILED buffer.
SAIEVN_GET_PARAMETER_DONE	**saiGetParamRecognizer** or **saiGetParamSynthesizer** request was processed. The result of the request is returned in the statusCode field of the event buffer. Returns additional information in a SAI_EVENT_COMMON buffer.
SAIEVN_LINK_CHANGED	A state change occurred at the speech server network connection. The current connection state result is indicated by the isUp field of the SAI_EVENT_LINK_CHANGED event buffer.
SAIEVN_SET_PARAMETER_DONE	**saiSetParamRecognizer** or **saiSetParamSynthesizer** request was processed. The result of the request is returned in the statusCode field of the event buffer. Returns additional information in a SAI_EVENT_COMMON buffer.
SAIEVN_START_OF_SPEECH	Speech synthesizer process has begun.
SAIEVN_TTS_CONTROL_DONE	**saiControlSynthesizer** request was processed. The result of the request is returned in the statusCode field of the event buffer. Returns additional information in a SAI_EVENT_TTS_CONTROL buffer.
SAIEVN_TTS_NOTIFY_BARGE_IN_DONE	**saiNotifyBargeInToSynthesizer** request was processed. The result of the request is returned in the statusCode field of the event buffer. Returns additional information in a SAI_EVENT_TTS_NOTIFY_BARGE_IN buffer.

Event name	Description
SAIEVN_TTS_PAUSE_DONE	**saiPauseSynthesizer** request was processed. The result of the request is returned in the statusCode field of the event buffer. Returns additional information in a SAI_EVENT_TTS_PAUSE buffer.
SAIEVN_TTS_RESUME_DONE	**saiPauseSynthesizer** request was processed. The result of the request is returned in the statusCode field of the event buffer. Returns additional information in a SAI_EVENT_TTS_RESUME buffer.
SAIEVN_TTS_SPEAK_ACKNOWLEDGED	**saiPauseSynthesizer** request was accepted and is being processed. The status code is indicated in the statusCode field of the event buffer. Returns additional information in a SAI_EVENT_COMMON buffer.
SAIEVN_TTS_SPEAK_DONE	When a speech server processes **saiSpeakSynthesizer**, this event indicates that the request is completed. The completion cause is returned in the completionCause field of the event buffer. Returns additional information in a SAI_EVENT_TTS_SPEAK_COMPLETE buffer.
SAIEVN_TTS_SPEAK_MARKER_REACHED	Unsolicited event that may be returned by the synthesizer when markers are included in the speech data specified with **saiSpeakSynthesizer**. The marker is returned in the event buffer. Retruns additional information in a SAI_EVENT_TTS_MARKER buffer.
SAIEVN_TTS_STOP_DONE	**saiStopSynthesizer** request was processed. The result of the request is returned in the statusCode field of the event buffer. Returns additional information in a SAI_EVENT_TTS_STOP buffer. Returns additional information in a SAI_EVENT_COMMON buffer.

Numerical listing

The following table numerically lists the USAI events:

Hex	Decimal	Event name
0x322001	3284993	SAIEVN_ENGINE_CREATION_DONE
0x322002	3284994	SAIEVN_ENGINE_RELEASE_DONE
0x322003	3284995	SAIEVN_GET_PARAMETER_DONE
0x322004	3284996	SAIEVN_SET_PARAMETER_DONE
0x322005	3284997	SAIEVN_ASR_DEFINE_GRAMMAR_DONE
0x322006	3284998	SAIEVN_ASR_RECOGNITION_STARTED
0x322007	3284999	SAIEVN_ASR_START_OF_SPEECH
0x322008	3285000	SAIEVN_ASR_RECOGNITION_DONE
0x322009	3285001	SAIEVN_ASR_STOP_DONE
0x32200A	3285002	SAIEVN_ASR_GET_RESULT_DONE
0x32200B	3285003	SAIEVN_ASR_START_TIMER_DONE
0x32200C	3285004	SAIEVN_TTS_SPEAK_ACKNOWLEDGED
0x32200D	3285005	SAIEVN_TTS_SPEAK_MARKER_REACHED
0x32200E	3285006	SAIEVN_TTS_SPEAK_DONE
0x32200F	3285007	SAIEVN_TTS_STOP_DONE
0x322010	3285008	SAIEVN_TTS_PAUSE_DONE
0x322011	3285009	SAIEVN_TTS_RESUME_DONE
0x322012	3285010	SAIEVN_TTS_CONTROL_DONE
0x322013	3285011	SAIEVN_TTS_NOTIFY_BARGE_IN_DONE
0x322014	3285012	SAIEVN_ENGINE_FAILED

Event information buffers

USAI events can return additional information in the following event buffers:

- SAI_EVENT_ASR_DEFINE_GRAMMAR
- SAI_EVENT_ASR_GET_RECOGNITION_RESULT
- SAI_EVENT_ASR_RECOGNITION_COMPLETE
- SAI_EVENT_ASR_START_OF_SPEECH
- SAI_EVENT_ASR_STOP
- SAI_EVENT_COMMON
- SAI_EVENT_ENGINE_CREATION
- SAI_EVENT_ENGINE_FAILED
- SAI_EVENT_ENGINE_RELEASE
- SAI_EVENT_LINK_CHANGED
- SAI_EVENT_TTS_CONTROL
- SAI_EVENT_TTS_MARKER
- SAI_EVENT_TTS_NOTIFY_BARGE_IN
- SAI_EVENT_TTS_PAUSE
- SAI_EVENT_TTS_RESUME
- SAI_EVENT_TTS_SPEAK_COMPLETE
- SAI_EVENT_TTS_STOP

SAI_EVENT_ASR_DEFINE_GRAMMAR

Buffer associated with the SAIEVN_ASR_DEFINE_GRAMMAR_DONE event that is returned when an ASR engine finishes downloading and compiling a specified grammar. The SAI_EVENT_ASR_DEFINE_GRAMMAR buffer consists of the following fields:

```
typedef struct
{
SAI_EVENT_HEADER                header;
SAI_STATUS_CODE                 statusCode;
SAI_RECOGNIZER_COMPLETION_CAUSE completionCause;
} SAI_EVENT_ASR_DEFINE_GRAMMAR;
```

where:

Field	Description
header	Event header.
statusCode	Status code that specifies result of the command by indicating the RTSP status code at the session establishment layer.
completionCause	Command completion cause

SAI_EVENT_ASR_GET_RECOGNITION_RESULT

Buffer associated with the SAIEVN_ASR_GET_RESULT_DONE event that is returned when a request to re-compute a recognition process completes. The SAI_EVENT_ASR_GET_RECOGNITION_RESULT buffer consists of the following fields:

```
typedef struct
{
SAI_EVENT_HEADER            header;
SAI_STATUS_CODE             statusCode;
char                        recognitionResult[1];
} SAI_EVENT_ASR_GET_RECOGNITION_RESULT;
```

where:

Field	Description
header	Event header.
statusCode	Status code that specifies result of the command by indicating the RTSP status code at the session establishment layer.
recognitionResult	Pointer to the beginning of a string that provides the recognition result. **Note:** recognitionResult is always the last element in this structure. Do not use the size of this structure to determine the size of the received structure because recognitionResult marks only the beginning of the recognition result.

SAI_EVENT_ASR_RECOGNITION_COMPLETE

Buffer associated with the SAIEVN_ASR_RECOGNITION_DONE event that is returned when the ASR engine completes (successfully or unsuccessfully) a recognition task.

If the completionCause value is SAI_RCC_NORMAL, then the recognitionResult value is valid. All media servers must support the W3C Natural Language Semantics Markup Language (NLSML) as the default standard for returning recognition results to the client, and must also support the MIME-type application/x-nlsml. The ASR context keeps the result and the audio waveform input of the recognition task until USAI receives the next recognition request .

The SAI_EVENT_ASR_RECOGNITION_COMPLETE buffer consists of the following fields:

```
typedef struct
{
SAI_EVENT_HEADER                header;
SAI_RECOGNIZER_COMPLETION_CAUSE completionCause;
char                            recognitionResult[1];
} SAI_EVENT_ASR_RECOGNITION_COMPLETE;
```

where:

Field	Description
header	Event header.
completionCause	Reason that the ASR engine completed the recognition task and sent the result.

recognitionResult	Result of the recognition task. The result format follows the nssml standard. For example:
	`Content-Type: application/x-nlsml`
	`Content-Length: 104`
	
	`<?xml version="1.0"?>`
	`<result grammar="http://theYesNoGrammar>`
	`<interpretation>`
	`<instance>`
	`<myApp:yes_no>`
	`<response>yes</response>`
	`</myApp:yes_no>`
	`</instance>`
	`<input>ok</input>`
	`</interpretation>`
	`</result>`
	
	Note: recognitionResult is always the last element of the structure. Do not use the size of this structure to determine the size of the received structure because recognitionResult marks only the beginning of the recognition result.

SAI_EVENT_ASR_START_OF_SPEECH

Buffer associated to the SAIEVN_ASR_START_OF_SPEECH event that is returned when the ASR engine detects a speech signal. The SAIEVN_ASR_START_OF_SPEECH event is useful in implementing kill-on-barge-in functionality when the synthesizer resource is in a different session than the recognizer resource and therefore is not aware of an incoming audio source. In these cases, the client must act as a proxy and invoke **saiNotifyBargeInToSynthesizer** to notify the synthesizer that a barge-in occurred.

The recognizer resource returns a unique proxy-sync-id in the event header, according to the proxySyncId parameter specified in **saiNotifyBargeInToSynthesizer**.

The SAI_EVENT_ASR_START_OF_SPEECH buffer consists of the following fields:

```
typedef struct
{
SAI_EVENT_HEADER         header;
SAI_PROXY_SYNC_ID        proxySyncId;
} SAI_EVENT_ASR_START_OF_SPEECH;
```

where:

Field	Description
header	Event header.
proxySyncId	A unique identifier generated by **saiNotifyBargeInToSynthesizer** and returned by the ASR engine. When the recognizer and synthesizer resources are part of the same session, they can work together to achieve quicker interaction and response times. In this case, the proxySyncId helps the resource that receives the event (the client acts as a proxy in transferring the event) to determine if the event was processed through a direct resource interaction.

SAI_EVENT_ASR_STOP

Buffer associated with the SAIEVN_ASR_STOP_DONE event that is returned when a request to stop a recognition process completes. The SAIEVN_ASR_STOP_DONE buffer consists of the following fields:

```
typedef struct
{
SAI_EVENT_HEADER            header;
SAI_STATUS_CODE             statusCode;
SAI_REQUEST_ID              stoppedReqId;
} SAI_EVENT_ASR_STOP;
```

where:

Field	Description
header	Event header.
statusCode	Status code that specifies result of the command by indicating the RTSP status code at the session establishment layer.
stoppedReqId	Request identifier of the recognition process associated with the stop command.

SAI_EVENT_COMMON

Buffer associated to the following events:

- SAIEVN_GET_PARAMETER_DONE
- SAIEVN_SET_PARAMETER_DONE
- SAIEVN_ASR_RECOGNITION_STARTED
- SAIEVN_ASR_START_TIMER_DONE
- SAIEVN_TTS_SPEAK_ACKNOWLEDGED
- SAIEVN_ENGINE_FAILED

The SAI_EVENT_COMMON buffer consists of the following fields:

```
typedef struct
{
SAI_EVENT_HEADER               header;
SAI_STATUS_CODE                statusCode;
} SAI_EVENT_COMMON;
```

where:

Field	Description
header	Event header.
statusCode	Status code that specifies the result of the command. Indicates the RTSP status code at the session establishment layer.

SAI_EVENT_ENGINE_CREATION

Buffer associated with the SAIEVN_ENGINE_CREATION_DONE event that indicates that USAI has either succeeded or failed in creating a TTS or ASR engine. The SAIEVN_ENGINE_CREATION buffer consists of the following fields:

```
typedef struct
{
SAI_EVENT_HEADER    header;
SAI_ENGINE_STATUS_CODE statusCode;
DWORD               enginePort;/
char                engineIp[SAI_MAX_IP_ADDRESS_LENGTH];
} SAI_EVENT_ENGINE_CREATION;
```

where:

Field	Description
header	Event header.
statusCode	Status code defining the result of the command. Indicates the RTSP status code at the session establishment layer.
enginePort	Engine RTP port.
engineIp	IP address of the speech server.

SAI_EVENT_ENGINE_FAILED

Buffer associated with the SAIEVN_ENGINE_FAILED event returned when a session connection problem occured at the TTS or ASR side. When an application receives an SAIEVN_ENGINE_FAILED event, it must invoke **saiReleaseSynthesizer** or **saiReleaseRecognizer** to release the associated TTS or ASR engine.

The SAI_EVENT_ENGINE_FAILED buffer consists of the following fields:

```
typedef struct
{
SAI_EVENT_HEADER           header;
SAI_ENGINE_STATUS_CODE     statusCode;
} SAI_EVENT_ENGINE_FAILED;
```

where:

Field	Description
header	Event header.
statusCode	Status code defining the result of the command. Indicates the RTSP status code at the session layer.

SAI_EVENT_ENGINE_RELEASE

Buffer associated with the SAIEVN_ENGINE_RELEASE_DONE event. The buffer indicates whether the ASR or TTS engine has been released. The SAIEVN_ENGINE_CREATION buffer consists of the following fields:

```
typedef struct
{
SAI_EVENT_HEADER              header;
SAI_ENGINE_STATUS_CODE        statusCode;
} SAI_EVENT_ENGINE_RELEASE;
```

where:

Field	Description
header	Event header.
statusCode	Status code that indicating result of the command by specifying the RTSP status code at the session establishment layer.

SAI_EVENT_LINK_CHANGED

Buffer associated with the SAIEVN_LINK_CHANGED event that indicates whether the speech server is up or down. The SAIEVN_LINK_CHANGED buffer consists of the following fields:

```
typedef struct
{
SAI_EVENT_HEADER              header;
BOOL                          isUp;
char                          serverName[SAI_MAX_SERVER_NAME];
} SAI_EVENT_LINK_CHANGED;
```

where:

Field	Description
header	Event header.
isUp	Boolean value determining whether the speech server went down or is returned to service. Valid values include the following: 1 = UP 0 = DOWN
serverName	String that indicates the server where the connection went down or returned to service.

SAI_EVENT_TTS_CONTROL

Buffer associated to the SAIEVN_TTS_CONTROL_DONE event that is returned when the a speech synthesis control request completes. The SAI_EVENT_TTS_CONTROL buffer consists of the following fields:

```
typedef struct
{
SAI_EVENT_HEADER           header;
SAI_STATUS_CODE            statusCode;
BOOL                       fSpeakRestart;
SAI_REQUEST_ID             reqList[ SAI_MAX_STOPREQUEST + 1 ]; // +1 for padding
} SAI_EVENT_TTS_CONTROL;
```

where:

Field	Description
header	Event header.
statusCode	Command status code.
fSpeakRestart	Indicate if a jump backwards to the beginning of a segment of speech data has occurred.
reqList	The list of active request identifiers associated with the control command.

SAI_EVENT_TTS_MARKER

Buffer associated with the unsolicited SAIEVN_TTS_SPEAK_MARKER_REACHED event that is returned when the TTS engine detects an occurrence of a specified marker. The SAI_EVENT_TTS_MARKER buffer consists of the following fields::

```
typedef struct
{
SAI_EVENT_HEADER            header;
SAI_MARKER                  markerStr;
} SAI_EVENT_TTS_MARKER;
```

where:

Field	Description
header	Event header.
markerStr	String that indicates the marker that was reached.

SAI_EVENT_TTS_NOTIFY_BARGE_IN

Buffer associated to the SAIEVN_TTS_NOTIFY_BARGE_IN_DONE event that is returned when a NOTIFY-BARGE-IN request completes. The SAI_EVENT_TTS_NOTIFY_BARGE_IN buffer consists of the following fields:

```
typedef struct
{
SAI_EVENT_HEADER          header;
SAI_STATUS_CODE           statusCode;
SAI_REQUEST_ID            reqList[ SAI_MAX_STOPREQUEST + 1 ]; // +1 for padding
} SAI_EVENT_TTS_NOTIFY_BARGE_IN;
```

where:

Field	Description
header	Event header.
statusCode	Command status code
reqList	Array of SAI_REQUEST_IDs associated with speech synthesis requests to be terminated when the TTS engine receives the NOTIFY-BARGE-IN command.

SAI_EVENT_TTS_PAUSE

Buffer associated to the SAIEVN_TTS_PAUSE_DONE event that is returned when a request to pause an active speech synthesis process completes. The SAI_EVENT_TTS_PAUSE buffer consists of the following fields:

```
typedef struct
{
SAI_EVENT_HEADER            header;
SAI_STATUS_CODE             statusCode;
SAI_REQUEST_ID              pausedReqId;        /* The paused reqId */
} SAI_EVENT_TTS_PAUSE;
```

where:

Field	Description
header	Event header.
statusCode	Command status code
pausedReqId	The request identifier of the paused speech synthesis process.

SAI_EVENT_TTS_RESUME

Buffer associated to the SAIEVN_TTS_RESUME_DONE event that is returned when the request to resume a paused speech synthesis process completes. The SAI_EVENT_TTS_RESUME buffer consists of the following fields:

```
typedef struct
{
SAI_EVENT_HEADER            header;
SAI_STATUS_CODE             statusCode;
SAI_REQUEST_ID              resumedReqId;
} SAI_EVENT_TTS_RESUME;
```

where:

Field	Description
header	Event header.
statusCode	Command status code
resumedReqId	The request identifier of the speech synthesis process to resume.

SAI_EVENT_TTS_SPEAK_COMPLETE

Buffer associated with the SAIEVN_TTS_SPEAK_DONE event that is returned when the TTS engine successfully or unsuccessfully completes a speech synthesis command. The SAI_EVENT_TTS_SPEAK_COMPLETE buffer consists of the following fields:

```
typedef struct
{
SAI_EVENT_HEADER            header;
SPEAK_COMPLETION_CAUSE      completionCause;
} SAI_EVENT_TTS_SPEAK_COMPLETE;
```

where:

Field	Description
header	Event header.
completionCause	String that indicates the completion cause. Refer to SAI_SPEAK_COMPLETION_CAUSE for a list of possible causes.

SAI_EVENT_TTS_STOP

Buffer associated to the SAIEVN_TTS_STOP_DONE event that is returned when a request to stop a list of speech synthesis tasks completes. The SAI_EVENT_TTS_STOP buffer consists of the following fields:

```
typedef struct
{
SAI_EVENT_HEADER          header;
SAI_STATUS_CODE           statusCode;
SAI_REQUEST_ID            reqList[ SAI_MAX_STOPREQUEST + 1 ]; // +1 for padding
} SAI_EVENT_TTS_STOP;
```

where:

Field	Description
header	Event header.
statusCode	Command status code
reqList	Array containing the request identifiers of all pending speech synthesis requests to be terminated by the stop command.

Universal Speech Access API constants

The Universal Speech Access API defines the following constants for defining synthesizer and recognizer parameters:

- SAI_AUDIO_FETCH_HINT
- SAI_ENGINE_STATUS_CODE
- SAI_FETCH_HINT
- SAI_GRAMMAR_TYPE
- SAI_METHOD_TYPE
- SAI_RECOGNIZER_COMPLETION_CAUSE
- SAI_REQUEST_STATE
- SAI_SPEAK_COMPLETION_CAUSE
- SAI_SPEAK_DATA_TYPE
- SAI_SPEECH_LANGUAGE_TYPE
- SAI_STATUS_CODE
- SAI_VOICE_GENDER
- Other constants

SAI_AUDIO_FETCH_HINT

Audio fetch hints value that defines whether or not the synthesizer can optimize speech by pre-fetching audio. (for example, when the application calls **saiTtsSetAudioFetchHint**):

Constant	Decimal	Description
SAI_AUDIO_FETCH_HINT_PREFETCH	0	Permit, but not require the platform pre-fetch audio.
SAI_AUDIO_FETCH_HINT_SAFE	1	Only fetch audio when it is needed, never before.
SAI_AUDIO_FETCH_HINT_STREAM	2	Allow the platform to stream audio fetches.
SAI_AUDIO_FETCH_HINT_LAST	3	Last entry in constant table.

SAI_ENGINE_STATUS_CODE

Status code enumeration resulting from a **saiCreateRecognizer** or **saiCreateSynthesizer** calls that indicates the success or failure of the request to establish a session or provides status information about the request. USAI can also return SAI_ENGINE_STATUS_CODE when a failure occurs in an open session, in which case the status code is provided in the SAI_EVENT_ENGINE_FAILED event buffer.

For more information about SAI_ENGINE_STATUS_CODE constants, refer to the RTSP specification.

Constant	Decimal
SAI_ENGINE_SUCCESS	200
SAI_ENGINE_CREATED	201
SAI_ENGINE_NOT_ENOUGH_MEMORY	250
SAI_ENGINE_MULTIPLE_CHOICES	300
SC_CSC_MOVED_PERMANENTLY	301
SAI_ENGINE_TEMPORARILY	302
SAI_ENGINE_OTHER	303
SAI_ENGINE_NOT_MODIFIED	304
SAI_ENGINE_USE_PROXI	305
SAI_ENGINE_BAD_REQUEST	400
SAI_ENGINE_UNAUTHORIZED	401
SAI_ENGINE_PAYMENT_REQUIRED	402
SAI_ENGINE_FORBIDDEN	403
SAI_ENGINE_FOUND	404
SAI_ENGINE_METHOD_NOT_ALLOWED	405
SAI_ENGINE_NOT_ACCEPTABLE	406
SAI_ENGINE_PROXI_AUTHENTICATION_REQUIRED	407
SAI_ENGINE_REQUEST_TIMEOUT	408
SAI_ENGINE_GONE	410
SAI_ENGINE_LENGTH_REQUIRED	411
SAI_ENGINE_PRECONDITION_FAILED	412
SAI_ENGINE_REQUEST_ENTITY_TOO_LARGE	413
SAI_ENGINE_REQUEST_URL_TOO_LARGE	414
SAI_ENGINE_UNSUPPORTED_MEDIA_TYPE	415
SAI_ENGINE_PARAMETER_NOT_UNDERSTOOD	451
SAI_ENGINE_CONFERENCE_NOT_FOUND	452

Constant	Decimal
SAI_ENGINE_NOT_ENOUGH_BANDWIDTH	453
SAI_ENGINE_SESSION_NOT_FOUND	454
SAI_ENGINE_METHOD_NOT_VALID_IN_THIS_STATE	455
SAI_ENGINE_HDR_FIELD_NOT_VALID_FOR_RESOURCE	456
SAI_ENGINE_INVALID_RANGE	457
SAI_ENGINE_PARAMETER_IS_READ_ONLY	458
SAI_ENGINE_AGGREGATE_OPERATION_NOT_ALLOWED	459
SAI_ENGINE_ONLY_AGGREGATE_OPERATION_ALLOWED	460
SAI_ENGINE_UNSUPPORTED_TRANSPORT	461
SAI_ENGINE_DESTINATION_UNREACHABLE	462
SAI_ENGINE_INTERNAL_SERVER_ERROR	500
SAI_ENGINE_NOT_IMPLEMENTED	501
SAI_ENGINE_BAD_GATEWAY	502
SAI_ENGINE_SERVICE_UNAVAILABLE	503
SAI_ENGINE_GATEWAY_TIMEOUT	504
SAI_ENGINE_RTSP_VERSION_NOT_SUPPORTED	505
SAI_ENGINE_OPTION_NOT_SUPPORT	551

SAI_FETCH_HINT

Fetch hints value that defines when the synthesizer retrieves content from the server (for example, when the application calls **saiTtsSetFetchHint**). This constant is used when setting URI access properties if the speech must fetch documents or other resources such as speech markup or audio files:

Constant	Decimal	Description
SAI_FETCH_HINT_PREFETCH	0	A file may be downloaded when the request is received.
SAI_FETCH_HINT_SAFE	1	A file that should only be downloaded when actually needed.
SAI_FETCH_HINT_LAST	2	Last entry in the constant table.

SAI_GRAMMAR_TYPE

Define the type of user-defined grammar:

Constant	Decimal	Description
SAI_GRAMMAR_TYPE_URL	0	Grammar includes URL that will be loaded by the speech engine.
SAI_GRAMMAR_TYPE_XML	1	Grammar is formatted following XML rules
SAI_GRAMMAR_TYPE_GRAMMAR_LIST	2	Grammar is actually a table of grammar.
SAI_GRAMMAR_TYPE_LAST	3	Last entry in the constant table.

SAI_METHOD_TYPE

Parameter defining the methodType field property when specific USAI functions are invoked:

Constant	Decimal	USAI function invoked....
SAI_METHOD_TTS_GET_PARAM	1	**saiGetParamSynthesizer**
SAI_METHOD_TTS_SET_PARAM	2	**saiSetParamSynthesizer**
SAI_METHOD_TTS_SPEAK	3	**saiSpeakSynthesizer**
SAI_METHOD_TTS_STOP	4	**saiStopSynthesizer**
SAI_METHOD_TTS_PAUSE	5	**saiPauseSynthesizer**
SAI_METHOD_TTS_RESUME	6	**saiResumeSynthesizer**
SAI_METHOD_TTS_BARGE_IN_OCCURRED	7	**saiNotifyBargeInToSynthesizer**
SAI_METHOD_TTS_CONTROL	8	**saiControlSynthesizer**
SAI_METHOD_ASR_GET_PARAM	9	**saiGetParamRecognizer**
SAI_METHOD_ASR_SET_PARAM	10	**saiSetParamRecognizer**
SAI_METHOD_ASR_DEFINE_GRAMMAR	11	**saiDefineGrammarRecognizer**
SAI_METHOD_ASR_RECOGNIZE	12	**saiStartRecognizer**
SAI_METHOD_ASR_GET_RESULT	13	**saiGetResultRecognizer**
SAI_METHOD_ASR_RECOGNITION_START_TIMERS	14	**saiStartTimerRecognizer**
SAI_METHOD_ASR_STOP	15	**saiStopRecognizer**
SAI_METHOD_LAST	16	Last entry in the constant table.

SAI_RECOGNIZER_COMPLETION_CAUSE

Speech recognizer cause enumeration:

Constant	Decimal	Description
SAI_RCC_NORMAL	0	Speech recognizer request completed with a match, or DEFINE-GRAMMAR succeeded in downloading and compiling the grammar.
SAI_RCC_NO_MATCH	1	Speech recognizer request completed, but no match was found.
SAI_RCC_INPUT_TIMEOUT	2	Speech recognizer request completed without a match due to a no-input-timeout.
SAI_RCC_RECOGNITION_TIMEOUT	3	Speech recognizer request completed without a match due to a recognition-timeout.
SAI_RCC_GRAMMAR_LOAD_FAILURE	4	Speech recognizer request failed due to a grammar load failure.
SAI_RCC_GRAMMAR_COMPILE_FAILURE	5	Speech recognizer request failed due to grammar compilation failure.
SAI_RCC_ERROR	6	Speech recognizer request terminated prematurely due to a recognizer error.
SAI_RCC_SPEECH_TO_EARLY	7	Speech recognizer request terminated because speech was too early.
SAI_RCC_TOO_MUCH_SPEECH_TIMEOUT	8	Speech recognizer request terminated because the supplied was too long.
SAI_RCC_URL_FAILURE	9	There was a failure accessing a URL.
SAI_RCC_LANGUAGE_UNSUPPORTED	10	Language not supported.
SAI_RCC_LAST	11	Last entry in the constant table.

SAI_REQUEST_STATE

Request state code enumeration:

Constant	Decimal	Description
SAI_REQUEST_COMPLETE	0	Request is completed
SAI_REQUEST_INPROGRESS	1	Request is in progress
SAI_REQUEST _PENDING	2	Request is pending
SAI_REQUEST_STATE_LAST	3	Last entry in the constant table.

SAI_SPEAK_COMPLETION_CAUSE

Speech synthesizer cause enumeration:

Constant	Decimal	Description
SAI_SCC_NORMAL	0	Speech synthesizer request completed normally,
SAI_SCC_BARGE_IN	1	Speech synthesizer request completed because of a barge-in detection.
SAI_SCC_PARSE_FAILURE	2	Speech synthesizer request terminated because of a failure to parse the speech markup text.
SAI_SCC_URL_FAILURE	3	Speech synthesizer request terminated because, access to one of the URLs failed.
SAI_SCC_ERROR	4	Speech synthesizer request terminated prematurely due to synthesizer error.
SAI_SCC_LANGUAGE_UNSUPPORTED	5	Speech synthesizer request terminated because of a language not supported by the synthesizer.
SAI_SCC_LAST	6	Last entry in the constant table.

SAI_SPEAK_DATA_TYPE

Constant	Decimal	Description
SAI_SPEAK_DATA_TYPE_URL	0	The data sent with **saiSpeakSynthesizer** is a URL.
SAI_SPEAK_DATA_TYPE_XML	1	The data sent with **saiSpeakSynthesizer** is in XML format.
SAI_SPEAK_DATA_TYPE_PLAIN_TEXT	2	The data sent with **saiSpeakSynthesizer** is a non-formatted plain text.
SAI_SPEAK_DATA_TYPE_DATA_LIST	3	The data sent with **saiSpeakSynthesizer** contains a list of speak rules.
SAI_SPEAK_DATA_TYPE_LAST	4	Last entry in the constant table.

SAI_SPEECH_LANGUAGE_TYPE

Predefined language types for the speech synthesizer to use. If the language is not predefined here, you must use a user-defined type. Refer to the speech vendor documentation for more information about supported languages.

Constant	Decimal	Description
SAI_SPEECH_LANGUAGE_TYPE_US_ENGLISH	0	US English
SAI_SPEECH_LANGUAGE_TYPE_UK_ENGLISH	1	UK English
SAI_SPEECH_LANGUAGE_TYPE_DUTCH	2	Dutch
SAI_SPEECH_LANGUAGE_TYPE_FRENCH	3	French
SAI_SPEECH_LANGUAGE_TYPE_GERMAN	4	German
SAI_SPEECH_LANGUAGE_TYPE_ITALIAN	5	Italian
SAI_SPEECH_LANGUAGE_TYPE_NORWEGIAN	6	Norwegian
SAI_SPEECH_LANGUAGE_TYPE_SPANISH_EUROPEAN	7	European Spanish
SAI_SPEECH_LANGUAGE_TYPE_SWEDISH	8	Swedish
SAI_SPEECH_LANGUAGE_TYPE_USER_DEFINED	9	User-defined language
SAI_SPEECH_LANGUAGE_TYPE_LAST	10	Last entry in the constant table.

SAI_STATUS_CODE

Result of a specific resource control command representing the success or failure or other status of the request.

Constant	Decimal	Description
SAI_SC_SUCCESS	200	Request has been executed successfully.
SAI_SC_SUCCESS_IGNORED	201	Success with some optional parameters ignored.
SAI_SC_FAILURE_MASK	400	Mask determining failure results.
SAI_SC_METHOD_NOT_ALLOWED	401	Request (command) is not supported by the speech server.
SAI_SC_METHOD_NOT_VALID_IN_THIS_STATE	402	Speech engine is not ready or not in the appropriate state to accept this request.
SAI_SC_UNSUPPORTED_PARAMETER	403	At least one command parameter is not supported.
SAI_SC_ILLEGAL_PARAMETER_VALUE	404	Parameter is supported but contains an illegal value.
SAI_SC_NOT_FOUND	405	Resource URI not initialized or doesn't exist
SAI_SC_MANDATORY_PARAMETER_MISSING	406	A mandatory parameter is missing.
SAI_SC_OPERATION_FAILED	407	Grammar compilation failed in the recognizer. The cause is indicated in the completion cause field.
SAI_SC_UNSUPPORTED_MESSAGE_ENTITY	408	An unrecognized or unsupported message entity was sent to the speech server.
SAI_SC_UNSUPPORTED_PARAMETER_VALUE	409	One of the parameters contains a non-supported value.
SAI_SC_RESOURCE_SPECIFIC_FAILURE_START	421	Beginning of the vendor specific failure code range. Refer to the vendor documentation for more information.
SAI_SC_RESOURCE_SPECIFIC_FAILURE_END	499	End of the vendor specific failure code range. Refer to the vendor documentation for more information.

SAI_VOICE_GENDER

Type of voice gender the Synthesizer uses when defining voice profiles with the SAI_VOICE structure:

Constant	Decimal	Description
SAI_VOICE_GENDER_NEUTRAL	0	Neutral voice.
SAI_VOICE_GENDER_MALE	1	Male voice.
SAI_VOICE_GENDER_FEMALE	2	Female voice.
SAI_VOICE_GENDER_LAST	3	Last entry in the constant table.

Other constants

The following table lists other constants used by the SAI service:

Constant	Decimal	Description
SAI_MAX_STOPREQUEST	10	Maximum number of request identifiers that can be contained in the reqList[] field of the extInfo buffer received with the returned event in **saiStopRecognizer** or **saiStopSynthesizer**.
SAI_MAX_IP_ADDRESS_LENGTH	32	Maximum string length when defining an IP address.
SAI_MAX_SERVER_NAME	256	Maximum string length for a server name.
SAI_MAX_URLLENGTH	256	Maximum string length when defining a URL address.
SAI_MAX_LOGGINGTAGLENGTH	256	Maximum string length when defining a logging tag.
SAI_MAX_STRINGLENGTH	256	Maximum string length when setting various string type parameters.
SAI_MAX_TAGLENGTH	256	Maximum string length when setting various tag type parameters.

Universal Speech Access API structures

The Universal Speech Access API defines the following structures for specifying synthesizer and recognizer properties:

- SAI_PROSODY
- SAI_RTP_ENDPOINT
- SAI_SPEECH_LANGUAGE
- SAI_VENDOR_SPECIFIC
- SAI_VOICE

SAI_PROSODY

SAI_PROSODY defines speaker voice qualities such as pitch, duration, and volume when invoking **saiTtsGetProsody** and **saiTtsSetProsody**:

```
typedef struct
{
INT32              pitch;
INT32              contour;
INT32              range;
INT32              rate;
INT32              duration;
INT32              volume;
}
SAI_PROSODY;
```

The SAI_PROSODY structure contains the following parameters:

Parameter	Description
pitch	Baseline pitch for the spoken text.
contour	Actual pitch contour for the spoken text.
range	Pitch range (variability) for the spoken text.
rate	Speaking rate for the spoken text.
duration	Value in seconds or milliseconds that sets the amount of time to take to read the text.
volume	Value from 0 through 100 that sets the volume for the spoken text.

Parameter values must follow the W3C Speech Synthesis Markup Language specification (SSML). For more information, refer to the SSML specification.

Before setting these voice elements, the application must retrieve the current values using **saiTtsGetProsody**. The application can then modify prosody parameters in the SAI_PROSODY structure with **saiTtsSetProsody**.

SAI_RTP_ENDPOINT

SAI_RTP_ENDPOINT defines a Fusion RTP endpoint. Reference this structure when invoking **saiCreateSynthesizer** and **saiCreateRecognizer**:

```
typedef struct
{
DWORD                  port;
SAI_URL                ipAddress;
}
SAI_RTP_ENDPOINT;
```

The SAI_RTP_ENDPOINT structure contains the following parameters:

Parameter	Description
port	RTP port number associated with a Fusion RTP endpoint.
ipAddress	IP address associated with a Fusion RTP endpoint.

SAI_SPEECH_LANGUAGE

SAI_SPEECH_LANGUAGE defines the recognizer or synthesizer language retrieved with **saiAsrGetSpeechLanguage** and **saiTtsGetSpeechLanguage**, or set with **saiAsrSetSpeechLanguage** and **saiTtsSetSpeechLanguage**:

```
typedef struct
{
SAI_SPEECH_LANGUAGE             language;
SAI_SPEECH_LANGUAGE_STRING      languageString;
}
SAI_SPEECH_LANGUAGE;
```

The SAI_SPEECH_LANGUAGE structure contains the following parameters:

Parameter	Description
pitch	Baseline pitch for the spoken text.
contour	Actual pitch contour for the spoken text.
range	Pitch range (variability) for the spoken text.
rate	Speaking rate for the spoken text.
duration	Value in seconds or milliseconds that sets the amount of time to take to read the text.
volume	Value from 0 through 100 that sets the volume for the spoken text.

The SAI_SPEECH_LANGUAGE parameter specifies a decimal value associated with a predefined language for the speech synthesizer to use. The SAI_SPEECH_LANGUAGE_STRING parameter specifies the character string associated with the language to use. The following table shows predefined language types that the speech synthesizer can use:

If the language is not predefined, the application must specify a user-defined language type. For more information about supported languages, refer to the speech vendor documentation.

SAI_VENDOR_SPECIFIC

Structure that defines vendor specific parameters.

```
typedef struct
{
char                vendorPairName[SAI_MAX_SREINGLENGTH];
char                vendorPairValue[SAI_MAX_SREINGLENGTH];
}SAI_VENDOR_SPECIFIC;
```

The SAI_VENDOR_SPECIFIC structure contains the following parameters:

Parameter	Description
vendorPairName	Keyword for a vendor-specific parameter.
vendorPairValue	Value for a vendor-specific parameter.

SAI_VOICE

SAI_VOICE defines part of the speech synthesizer voice profile used when invoking **saiTtsGetVoice** and **saiTtsSetVoice**:

```
typedef struct
{
    SAI_VOICE_GENDER            gender;
    INT32                       age;
    char                        variant[SAI_MAX_STRINGLENGTH];
    char                        name[SAI_MAX_STRINGLENGTH];
}
SAI_VOICE;
```

The SAI_VOICE structure contains the following parameters:

Parameter	Description
gender	Value of SAI_VOICE_GENDER constant. This can be neutral, female, or male.
age	Defines the age of the speaker's voice in years.
variant	Defines a variant in the speaker's voice according to the speech vendor's specification.
name	Defines a speaker's voice pre-packaged name. Refer to the speech vendor documentation for a list of predefined voice packages.

Note: Parameter values must follow the W3C Speech Synthesis Markup Language specification (SSML). For more information, refer to the SSML specification.

Before setting voice profile parameters, the application must retrieve the current values with **saiTtsGetVoice**. The application can then modify voice profile parameters in the SAI_VOICE structure and invoke **saiTtsSetVoice**.

Index