Vocalware API Reference

Table of Contents

Introduction

The Vocalware API enables you to use our cloud based Text-To-Speech service, to generate & play audio in real-time within your online application. By the term "online application" we refer to *any online program*, including: web pages, or native code apps on either desktop, server or mobile device. The only requirement is that your application has access to an internet connection fast enough to stream 48kbps audio.

The Vocalware API allows you to generate audio and control audio playback. **The API comes in two flavors: JavaScript/HTML5, and HTTP-REST** - so it can be easily incorporated into any application. Whether your application runs in-browser or standalone, on mobile, desktop or server - one of our API flavors will work for you.

> *Note: A 3rd API 'flavor' - Actionscript (Flash) - has been deprecated and is no longer offered. For backwards compatibility we will continue to support existing customers using the ActionScript API as long as there is active interest. For the same reason this document still contains the information needed to support ActionScript developers.*

The Vocalware API supports TTS in over 20 languages, with several voices available in most. The API allows you to specify the language and voice to use, as well as optional audio effects such as pitch, echo, etc.

Select the API Flavor to Use

The API comes in two flavors:

1. JavaScript/HTML5 - also referred to as the JavaScript API
2. HTTP-REST - also referred to as the HTTP API

A 3rd flavor, for Flash developers, has been deprecated but is still supported for backwards compatibility -

3. ActionScript - both AS2 and AS3 are supported

To proceed, you should first identify the section of this document that refers to the API flavor you plan to use. The JavaScript and ActionScript APIs are similar, and are both covered by the first section of this guide. The HTTP API is covered by the second section.

How to select the API flavor that's right for you? Here are several rules of thumb:

- in your web pages – use the JavaScript API (supports mobile browsers as well)
- in your standalone (out of browser) app, including mobile app - use the HTTP API
- on your server – use the HTTP API

Note: Whether mobile, desktop or server, the decision boils down to whether your intended use is within a web browser or not. If it is, use the JS API. If it is not – use the HTTP API.

Additional Resources

If you have any questions, or run into difficulty when trying to use any of our APIs, please check out our support pages, where you will be able to access:

- Frequently Asked Questions covering a large number of issues.
- API examples, including full source code, covering both APIs and each of the API functions listed here.
- Send a note to our support team.

The JavaScript & ActionScript APIs

Introduction

The API function calls for JavaScript and ActionScript are identical in syntax and functionality. Both JavaScript and ActionScript APIs operate by way of an invisible client side code object ("Agent"), that your web page or Flash application loads & can then access via the API functions.

> ***Note***: This works transparently on both Desktop and Mobile browsers, as the client side Agent code automatically adapts to the client platform's capabilities.

The API supports TTS audio generation as well as playback control functionality. The interface consists of a set of client side JavaScript or ActionScript calls, and does not require you to make any call to our servers, as the client side API encapsulates all interaction with the Vocalware servers.

> ***Tip:*** The simplest way to handle playback of the generated audio, is to control it via the documented API functions. Using these high level functions makes direct access to the audio data unnecessary in most cases. However, if you need low level access to the audio stream data, the API allows ActionScript data direct access through the "vw_audioStarted" callback (available only in AS3).

To get started with either JS or AS APIs you need to:

a. Create an AS or JS API object in your account's My APIs page. Copy the 'embed code' unique to your API object – and paste it into the BODY section of your page.
b. Specify in your 'Security Settings' page the domain (or several domains) in which this API is to operate.
c. Implement the vw_apiLoaded callback to receive notice that your API is ready.

Important caveats / pitfalls to avoid -

- Your embed code is specific to your account and for your protection will allow playback only from the domain(s) you specify. **Specifying a domain is mandatory. Your API will not function without it.**
- The "vw_apiLoaded" status callback is dispatched when the API is ready. It is therefore advisable to implement the "vw_apiLoaded" callback – and avoid calling any API function prior to receiving confirmation that loading has completed. **API functions work only after the API has completed loading.**

In the next sections you will find instructions and code examples explaining how to use your embed code as well as a listing of the API function calls.

Programming for Mobile

The JavaScript API operates transparently on 'desktop' as well as mobile browsers (the term 'desktop' is used throughout to refer to non-mobile client side environments, which include desktop and laptop computers of all types). This means that you need not do anything special in order to support mobile functionality in your web pages when using the Vocalware JavaScript API. That said, there are a couple of differences between mobile and desktop that you should be aware of.

The JavaScript API is fully compatible with all major mobile browsers – and with two exceptions will function in the same way within mobile browsers as it will on desktop browsers. One exception is with the function 'setPlayerVolume' – which does not have any effect in some mobile browsers – but there is no harm in making the call.

Another important difference is that on most mobile browsers (notably Safari which is the default browser on iOS), the first call to the API must be user driven (i.e. user clicks on a button). This restriction prevents the web page from automatically speaking to the viewer unprompted. Trying to do so will not cause an error – but will simply not work.

Note that your embed code must be placed within the BODY section of the HTML page, and will not work otherwise! See additional detail below in "Using your Embed Code".

Using your Embed Code

The embed code is a code segment unique to your account, or more specifically to an API Object within your account. Instructions and examples below explain how to incorporate the embed code.

JavaScript Instructions

Paste your embed code into your HTML page's BODY section. Needless to say that your page must have a BODY section to be able to fulfill this requirement… This instruction applies to mobile as well as non-mobile web pages.
The exact location within your HTML is not significant, though it is best not to include it within FORM brackets or other nested HTML structures.

Use the Javascript API functions defined below.

ActionScript 3 Instructions

1. Run Adobe Flash CS3 or higher.
2. Click Layer 1
3. If you do not see the Actions > Frame window label in the middle left of the screen, click Actions.
4. Add the following block of code into the Actions Frame window.

```
Security.allowDomain("content.oddcast.com");
var ldr:Loader;
var req:URLRequest;

var vw_player:MovieClip;
var _example_ui:MovieClip;

req = new URLRequest("EMBED_CODE");
ldr = new Loader();
ldr.contentLoaderInfo.addEventListener(Event.COMPLETE,completeHan
dler);
ldr.load(req);
addChild(ldr);

function completeHandler($ev:Event):void
{
   trace("EXAMPLE --- COMPLETE HANDLER "+$ev.target);
   vw_player = MovieClip(ldr.content);
}
```

5. Copy your embed code and Paste where you see EMBED_CODE above.

Note: In the HTML file make sure that allowScriptAccess' is set to 'always'.

6. Declare event listeners in the completeHandler function if call back functions will be used, for example:

```
vw_player.addEventListener("vw_apiLoaded", vw_apiLoaded);
```

ActionScript 2 Instructions

1. Run Adobe Flash MX or higher.
2. From the top level menu, choose Insert->New Symbol->Ok
3. Click on Scene 1
4. The Scene window appears as a blank white rectangle in the center of the Adobe Flash screen.
5. Drag Symbol 1 from the Library window to the upper left corner of the Scene window. Replace with instance_name.
6. Click Layer 1
7. If you do not see the Actions > Frame window label in the middle left of the screen, click Actions.
8. Copy your embed code and Paste into the Actions Frame window.
9. The line below should appear in the Actions for the first Frame of the first Scene of your movie:

```
System.security.allowDomain("vhost.oddcast.com", "vhss-
a.oddcast.com", "vhss-c.oddcast.com", "vhss-d.oddcast.com");
```

10. The line below should appear in the Actions for the Scene. Replace instance_name with the name of the instance name you entered for the symbol you inserted, and replace EMBED_CODE with your embed code.

    instance_name.loadMovie("EMBED_CODE");

11. Declare all callback functions that you wish to use, for example:

```
function vw_apiLoaded(){
        //any commands that should be triggered here;
}
```

Playback Control Functions

All functions are available to both JS and AS APIs. The syntax of the functions is the same in both.

sayText (txt,voice,lang,engine,[effect], [effLevel])

Real-time (dynamic) Text-To-Speech (TTS).

Note: This function will work only within a licensed domain for the account. Domain specific licensing is a security measure. If playback is attempted within a domain that is not specifically licensed for the account, this call will generate an error.

Arguments:

`txt` Required. String - The text to speak. Most languages are limited to 900 characters. The exceptions are Chinese & Japanese which are limited to 225 characters. A longer text string will be truncated.

`voice` Required. Integer – Voice ID, as listed in Appendix A.

`lang` Required. Integer – Language ID, as listed in Appendix A.

`engine` Required. Integer – Voice Family ID. See languages and voices listed listed in Appendix A.

`effect` Optional. Character. Audio effect – one of:

- "D" – Duration levels: -3, -2, -1, 1, 2, 3
- "P" – Pitch levels: -3, -2, -1, 1, 2, 3
- "S" – Speed levels: -3, -2, -1, 1, 2, 3
- "R" – Robotic:
  - Bullhorn level: 3 (note: levels 1 and 2 are deprecated)
- "T" – Time:
  - Echo level: 1
  - Reverb level: 2
  - Flanger level: 3
  - Phase level: 4
- "W" – Whisper levels: 1, 2, 3

`effLevel` Optional. Integer. Effect level must be provided if effect is provided.

Examples:

```
sayText('Hello World',1,1,1)
sayText('Hello World',1,1,1,'S',-2)
```

setPlayerVolume (level)

Set playback volume, or mute the audio.

Arguments:

`level`	Required. Integer (0-10) – Default = 7. a value from 0 to 10; 0 is equivalent to mute, 1 is softest, 10 is loudest.

Example:

```
setPlayerVolume(10)
```

Note:

- Setting the volume to 0, does not stop playback or the audio stream. It only affects the audio volume. To stop playback, use the function stopSpeech(). To pause playback, use the function freezeToggle().
- Calling this function has no effect on some mobile browsers.

---

stopSpeech ()

Stop audio playback in progress. If audio is not currently playing, stopSpeech has no effect (i.e. it does not prevent speech that has not yet begun).

Arguments:

None.

Example:

```
stopSpeech()
```

---

freezeToggle ()

Toggle between the pause and play states. If playback is in progress, it is paused. If playback is paused, it is resumed from the point it was paused.

Arguments:

None.

Example:

```
freezeToggle()
```

---

setStatus (interruptMode,progressInterval,reserved1,reserved2)

Set several status values which govern various aspects of playback.

Arguments:

`interruptMode`

Required. Integer (0/1) – Default = 0.
If set to 0 consecutive audio playback function calls (sayText) are queued for consecutive playback.
If set to 1 current audio is interrupted when sayText is called.

`progressInterval`

Required. Non-negative Integer – Default = 0.
The audio progress interval value controls progress callbacks which take place during playback. The callback function

vw_audioProgress(percent_played)

is called during playback if the value of 'progressInterval' is non-zero. The non-zero value determines the frequency of the call.
The value must be an integer greater than or equal to 0. When greater than 0, the callback "vw_audioProgress(percent_played)" is triggered at the frequency specified by the number (in seconds). The callback returns the percent of the current audio that has played. Callbacks will continue for all subsequent audios played once this field is set. Set back to 0 for the callbacks to cease.

`reserved1`

Required. Integer. Set to 0.

`reserved2`

Required. Integer. Set to 0.

Example:

```
setStatus(1,0,0,0)
```

Status Callback Functions

Callback Functions enable coordination between playback and your page or application.

Callback functions are supported in both Flash movies (ActionScript) and HTML pages (JavaScript). The syntax of the functions is the same, though the method of setting them up is different - please see below.

Embedding in an HTML page:

Events during playback trigger calls to specific JavaScript functions in your page, if such functions exist. To take advantage of these calls you must **add the appropriate JavaScript functions to your page**. Note that you do not need to add callback functions which you do not intend to use.

Embedding in a Flash movie:

ActionScript 2

Events during playback trigger calls to specific ActionScript functions in your movie, if such functions exist. To take advantage of these calls you must **add the callback functions within your movie at the _parent level**. Note that you do not need to add callback functions which you do not intend to use.

ActionScript 3

To receive the status callbacks you need to register an event listener for each callback function. Here's an example of loading the content and registering as a listener for the "vw_talkStarted" event:

```
loader:Loader = new Loader();
loader.loaderContentInfo.addEventListener(Event.COMPLETE,
setListeners);
loader.load( /* your AS3 embed code here */ );
function setListeners():void
{
MovieClip(loader.content).addEventListener("vw_talkStarted",talkS
tartedHandler);
function talkStartedHandler():void{ trace("talk started"); }
}
```

vw_apiLoaded (apiID)

Triggered when the API is fully loaded. Use this callback to verify API is ready, prior to making any function calls.

Arguments:

`apiID` The id of the api being loaded.

Example - JavaScript & ActionScript2

```
function vw_apiLoaded(apiID){
      alert("the API is loaded");
}
```

Example - ActionScript3

```
MovieClip(loader.content).addEventListener("vw_apiLoaded",ap
iLoadedHandler);
function apiLoadedHandler(event:*):void{
      trace("api loaded. Id="+ event.data);
}
```

---

vw_audioProgress (percentPlayed)

Called during playback, if and only if the 'progressInterval' status is set.
vw_audioProgress is repeatedly called at regular intervals during playback. The intervals are determined according to the value of the 'progressInterval' status. See 'setStatus' API call for information about how to set this value.
This callback can be used to enable synchronization between playback and other events taking place at the same time. For example: highliting text segments, or visual elements on the page in coordination with speech playback.

Arguments

`percentPlayed`	A value between 0 and 100 which indicated the proportion of audio already played.

Example - JavaScript & ActionScript2

```
function vw_audioProgress(percentPlayed) {
}
```

Example - ActionScript3

```
MovieClip(loader.content).addEventListener("vw_audioProgress",audi
oProgHandler);
function audioProgHandler(event:*):void{
      trace("percent played: "+ event.data.percent);
}
```

---

vw_talkStarted ()

Triggered when the audio playback begins.

Example - JavaScript & ActionScript2

```
function vw_talkStarted(){
}
```

Example - ActionScript3

```
MovieClip(loader.content).addEventListener("vw_talkStarted",talkS
tartedHandler);
function talkStartedHandler(event:*):void{
      trace("talk started");
}
```

---

vw_talkEnded ()

Triggered when audio playback is done.

Example - JavaScript & ActionScript2

```
function vw_talkEnded(){
}
```

Example - ActionScript3

```
MovieClip(loader.content).addEventListener("vw_talkEnded",talkEnde
dHandler);
function talkEndedHandler ():void{
      trace("talk ended");
}
```

---

vw_audioStarted ()

Triggered when audio playback begins. Unlike vw_talkStarted() this event is fired for each audio playback in a sequence. In ActionScript3, the event contains a "data" property which provides direct references to the Sound object (event.data.sound) and the SoundChannel (event.data.sound_channel) to allow advanced control for as3 developers.

Example - JavaScript & ActionScript2

```
function vw_audioStarted(){
}
```

Example - ActionScript3

```
MovieClip(loader.content).addEventListener("vw_audioStarted",audio
StartedHandler);

function audioStartedHandler(event:*):void{
      var sound:Sound = event.data.sound;
      var sound_channel:SoundChannel = event.data.sound_channel;
      trace("audio started");
}
```

---

vw_audioEnded ()

Triggered when audio playback ends. Unlike talkEnded() this event is fired for each audio playback in a sequence.

Example - JavaScript & ActionScript2

```
function vw_audioEnded(){
}
```

Example - ActionScript3

```
MovieClip(loader.content).addEventListener("vw_audioEnded",audEndedHandler);
function audEndedHandler ():void{
      trace("audio ended");
}
```

The HTTP REST API

Note: This API supports both HTTP and HTTPS protocols. For simplicity only HTTP is documented. Where you see HTTP mentioned you may assume either HTTP or HTTPS can be used.

The HTTP GEN Request

This HTTP request supports either GET or POST parameter passing. The syntax example below describes only the HTTP GET request.

Syntax Example:

http://www.vocalware.com/tts/gen.php?EID=2&LID=1&VID=2&TXT=Test+Message&EXT=mp3&FX_TYPE=p&FX_LEVEL=1&ACC=8879&API=283475&SESSION=50cdbf7c4e5d117dcb2efff424e10054&HTTP_ERR=1&CS=ec1e9c6249980c208186bd64ce8ce6e1

Note: BOLD parameters are required

Parameters	Description
EID	Engine Id.
LID	Language Id.
VID	Voice Id.
TXT	Text to be used for audio creation (URL Encoded)
EXT	swf or mp3. Default is mp3
FX_TYPE	Sound effect type. Default is empty (no effect)
FX_LEVEL	Sound effect level. Default is empty (no effect)
ACC	Account id
API	API id
SESSION	Used to verify the session (see Session Verification section)
HTTP_ERR	Optionally use HTTP header status codes to return success or error. Values: 0 – do not use HTTP codes (default) 1 – use HTTP codes
CS	Checksum – implemented as an md5 of all above parameter and your secret phrase

CS = md5 (**EID** + **LID** + **VID** + **TXT** + EXT + FX_TYPE + FX_LEVEL + **ACC** + **API** + SESSION + HTTP_ERR + **SECRET PHRASE**)

Return values:

In case of success, mp3 or swf binary stream is returned. The audio data is single channel (mono), has a 22Khz sample rate and is encoded at a 48Kbps bitrate.

In case of failure, an error code and message are returned. See HTTP Error Codes sub-section for more information.

Session Verification

Session verification is an optional feature designed to protect your account. Here's how it works:

- When your application makes an HTTP GEN request, and if the checksum proves to be authentic, we call a predefined URL on your servers (the "Callback URL").
- You specify the Callback URL for us to use as part of your Vocalware account security settings.
- The call is an HTTP POST request, with two parameters – your account ID and the session ID you provided when making the GEN request.
- When we call you – you may authorize the session, or reject it.
- If the Callback URL for the account is not setup, or if the Session parameter is not provided, then no callback attempt is made.
- If the Callback URL for the account is not setup, checksum is calculated without the Session parameter even if present.
- Note: we cache your responses. Subsequent GEN calls that provide the same session ID will not always generate a callback.

Why use session verification?
If your GEN requests originate from your server, there is no need to setup session verification. But if you are making GEN requests from a client application (i.e. a web page) – then session verification is highly advisable to secure your account.

Verification Syntax:
POST request to account Callback URL.

Parameter	Description
ACC	Account id
SESSION	Provided session id

Return Values	Description
1	SESSION is valid
0	Error – invalid session

Example

This example page demonstrates how to put together the HTTP GEN request:
http://www.vocalware.com/support/rest-api

The Create API Request

This call enables you to programatically create new APIs in your account. This can be useful for setting up individual users or clients to be able to track their usage via Vocalware analytics. The effect of this call is the same as logging into your account and manually creating a new API.
This HTTP request supports either GET or POST parameter passing. The syntax example below describes only the HTTP GET request.

Syntax Example:

http://www.vocalware.com/tts/createapi.php?ACC=8879&APIName=abcd&APIType=h&CS=ec1e9c6249980c208186bd64ce8ce6e1

Note: BOLD parameters are required

Parameters	**Description**
ACC	Account id
APIName	Name of new API to be created. String. Under 20 characters.
APIType	Type of new API to be created. "j" - Javascript; "h" - HTTP.
CS	Checksum – implemented as an md5 of all above parameter and you secret phrase

CS = md5 (**ACC** + **APIName** + **APIType** + **SECRET PHRASE**)

Return values:
In case of success -
if Type = j - Embed code for new API is returned.
if Type = h - ID of new API is returned.

In case of failure, an error code and message are returned. See HTTP Error Codes sub-section for more information.

The Get Stream Balance Request

This call enables you to retrieve the current stream balance from your Vocalware account. Note that the information is not real-time but updated about every 10 minutes, This HTTP request supports either GET or POST parameter passing. The syntax example below describes only the HTTP GET request.

Syntax Example:

http://www.vocalware.com/tts/getbalance.php?ACC=8879&CS=ec1e9c6249980c208186bd64ce8ce6e1

Note: BOLD parameters are required

Parameters	Description
ACC	Account id
CS	Checksum – implemented as an md5 of all above parameter and you secret phrase

CS = md5 (**ACC** + **SECRET PHRASE**)

Return values:
In case of success -

The number of available streams in your account is returned (integer).

In case of failure, an error code and message are returned. See HTTP Error Codes sub-section for more information.

HTTP Error Codes

Two types of error codes are used:

1. "In Stream" error codes are always returned in case of failure. The error code begins with the string "Error".
2. "HTTP Header" status codes are only returned if HTTP_ERR parameter is set to 1. Otherwise, code "200" will always be returned. By default, this parameter is set to 0 (for backward compatibility).

The following are the error codes returned for each type:

In-Stream Error Codes (always returned in case of error)		
Error Code	**Message**	**More Info**
100	No data found in request.	Missing all request parameters.
101	Missing Required Parameter	Missing EID

102	Missing Required Parameter	Missing LID
103	Missing Required Parameter	Missing VID
104	Missing Required Parameter	Missing TXT
105	Missing Required Parameter	Missing ACC
106	Missing Required Parameter	Missing API
107	Missing Required Parameter	Missing CS
108	Missing Required Parameter	Missing APIName
109	Missing Required Parameter	Missing APIType
201	Unknown account id	ACC failed verification
202	Invalid session	SESSION failed verification
203	Invalid checksum	Checksum failure
204	Authorization failure	Verification failed (General)
205	Inactive account	Inactive account
206	Invalid API	API ID not assigned to account
300	General error	General error
301	Too many TTS Requests	Too many TTS Requests
302	TTS Failed	TTS Failed
400	APS Failed	APS Failed

HTTP Header Status Codes (returned only if requested)	
Error Code	**Description**
200	Successful TTS request
400	Bad (malformed) request. Modify the request before re-submitting.
401	Unauthorized request.
503	The server is temporarily unable to fulfill the request. OK to re-submit.

Generating the Checksum

To calculate the checksum, concatenate all the parameters in the order they appear in this document and add your 'Secret Phrase', which you can find in the 'security settings' on your 'my APIs' page.

Apply the md5 one way function to the resulting string, to generate the checksum X, and append it to the parameter list as CS=X
The checksum is created in the following way:

CS = md5 (**EID** + **LID** + **VID** + **TXT** + EXT + FX_TYPE + FX_LEVEL + **ACC** + **API** + SESSION + HTTP_ERR + **SECRET PHRASE**)

Note:

- TXT value should not be encoded for checksum computation.
- Leading or trailing spaces should be trimmed from the TXT value

- Optional parameters are to be omitted when computing the checksum if missing, but included if present.

Checksum Generation PHP Code Example:

```
//Set optional values to empty if not given.
$ext   = isset($_POST['EXT']) &&
in_array(trim(strtolower($_POST['EXT'])), array('mp3','swf')) ?
trim(strtolower($_POST['EXT'])) : '';

$fxType = isset($_POST['FX_TYPE']) && strlen($_POST['FX_TYPE']) > 0 ?
$_POST['FX_TYPE'] : '';

$fxLevel= isset($_POST['FX_LEVEL']) && strlen($_POST['FX_LEVEL']) > 0 ?
$_POST['FX_LEVEL'] : '';

$httpErr= isset($_POST['HTTP_ERR']) && strlen($_POST['HTTP_ERR']) > 0 ?
$_POST['HTTP_ERR'] : '';

//Construct parameters.
$get =       'EID='.$_POST['EID']
       .'&LID='.$_POST['LID']
       .'&VID='.$_POST['VID']
       .'&TXT='.urlencode($_POST['TXT'])
       .'&EXT='.$ext
       .'&FX_TYPE='.$fxType
       .'&FX_LEVEL='.$fxLevel
       .'&ACC='.$_POST['ACC']
       .'&API='.$_POST['API']
       .'&SESSION='.$_POST['SESSION']
       .'&HTTP_ERR='.$httpErr;

//Construct checksum
$CS = md5($_POST['EID'].$_POST['LID'].$_POST['VID'].$_POST['TXT'].
$ext.$fxType.$fxLevel.$_POST['ACC']. $_POST['API'].$_POST['SESSION'].
$httpErr.$_POST['SECRET']);

//Construct full URL
$url = 'http://www.vocalware.com/tts/gen.php?' . $get . '&CS=' . $CS;
```

Appendix A: Languages and Voices

The following tables list Engine IDs, Language IDs and Voice IDs available for use with the Vocalware API.

Language	ID
Arabic	27
Catalan	5
Chinese	10
Danish	19
Dutch	11
English	1
Esperanto	31
Finnish	23
French	4
Galician	15
German	3
Greek	8
Italian	7
Japanese	12
Korean	13
Norwegian	20
Polish	14
Portuguese	6
Romanian	30
Russian	21
Spanish	2
Swedish	9
Turkish	16

Engine ID = 2

Language	*Lang. ID*	*Voice Name*	*Voice ID*	*Gender*	*Description*	*Expressive Cues**
English	1	Susan	1	F	US	√
English	1	Dave	2	M	US	√
English	1	Elizabeth	4	F	UK	√
English	1	Simon	5	M	UK	√
English	1	Catherine	6	F	UK	√
English	1	Allison	7	F	US	√
English	1	Steven	8	M	US	√
English	1	Alan	9	M	Australian	√
English	1	Grace	10	F	Australian	√
English	1	Veena	11	F	Indian	√
Spanish	2	Carmen	1	F	Castilian	√
Spanish	2	Juan	2	M	Castilian	√
Spanish	2	Francisca	3	F	Chilean	
Spanish	2	Diego	4	M	Argentine	
Spanish	2	Esperanza	5	F	Mexican	
Spanish	2	Jorge	6	M	Castilian	√
Spanish	2	Carlos	7	M	American	√
Spanish	2	Soledad	8	F	American	√
Spanish	2	Leonor	9	F	Castilian	√
Spanish	2	Ximena	10	F	American	√
German	3	Stefan	2	M		√
German	3	Katrin	3	F		√
French	4	Bernard	2	M	European	√
French	4	Jolie	3	F	European	√
French	4	Florence	4	F	European	√
French	4	Charlotte	5	F	Canadian	√
French	4	Olivier	6	M	Canadian	√
Catalan	5	Montserrat	1	F		√
Catalan	5	Jordi	2	M		√
Catalan	5	Empar	3	F	Valencian	√
Portuguese	6	Amalia	2	F	European	√
Portuguese	6	Eusebio	3	M	European	√
Italian	7	Paola	1	F		√
Italian	7	Silvana	2	F		√
Italian	7	Valentina	3	F		√
Italian	7	Luca	5	M		√
Italian	7	Marcello	6	M		
Italian	7	Roberto	7	M		
Italian	7	Matteo	8	M		√
Italian	7	Giulia	9	F		√
Italian	7	Federica	10	F		√
Greek	8	Afroditi	1	F		√

Greek	8	Nikos	3	M		√
Swedish	9	Annika	1	F		√
Swedish	9	Sven	2	M		√
Chinese	10	Linlin	1	F	Mandarin	
Chinese	10	Lisheng	2	F	Mandarin	
Dutch	11	Willem	1	M		√
Dutch	11	Saskia	2	F		√
Polish	14	Zosia	1	F		√
Polish	14	Krzysztof	2	M		√
Galician	15	Carmela	1	F		√
Turkish	16	Kerem	1	M		√
Turkish	16	Zeynep	2	F		√
Turkish	16	Selin	3	F		√
Danish	19	Frida	1	F		√
Danish	19	Magnus	2	M		√
Norwegian	20	Vilde	1	F		√
Norwegian	20	Henrik	2	M		√
Russian	21	Olga	1	F		√
Russian	21	Dmitri	2	M		√
Finnish	23	Milla	1	F		√
Finnish	23	Marko	2	M		√
Arabic	27	Tarik	1	M		√
Arabic	27	Laila	2	F		√
Romanian	30	Ioana	1	F		√
Esperanto	31	Ludoviko	1	M		√

* *Expressive Cues* are a set of special tags which you may use in your text to specify distinct non-verbal expressions, such as laughing, crying, sighing, coughing, etc. Expressive Cues can be used only with a subset of voices, as indicated above. For a complete list of Expressive Cue tags see separate documentation.

Engine ID = 3

Language	*Lang. ID*	*Voice Name*	*Voice ID*	*Gender*	*Description*
English	1	Kate	1	F	US
English	1	Paul	2	M	US
English	1	Julie	3	F	US
English	1	Bridget	4	F	UK
English	1	Hugh	5	M	UK
English	1	Ashley	6	F	US
English	1	James	7	M	US
English	1	Beth	8	F	US
Spanish	2	Violeta	1	F	Mexican
Spanish	2	Francisco	2	M	Mexican

Spanish	2	Gloria	3	F	Mexican
Spanish	2	Lola	4	F	Castilian
Spanish	2	Manuel	5	M	Castilian
German	3	Lena	1	F	
German	3	Tim	2	M	
French	4	Chloe	1	F	Canadian
French	4	Leo	2	M	Canadian
French	4	Roxane	3	F	
French	4	Louis	4	M	
Portuguese	6	Helena	1	F	Brazilian
Portuguese	6	Rafael	2	M	Brazilian
Italian	7	Elisa	1	F	
Italian	7	Roberto	2	M	
Chinese	10	Lily	1	F	Mandarin
Chinese	10	Hui	3	F	Mandarin
Chinese	10	Liang	4	M	Mandarin
Chinese	10	Qiang	5	M	Mandarin
Chinese	10	Kaho	6	M	Cantonese
Chinese	10	Kayan	7	F	Cantonese
Chinese	10	Yafrang	8	F	Taiwanese
Japanese	12	Show	2	M	
Japanese	12	Misaki	3	F	
Japanese	12	Sayaka	4	F	
Japanese	12	Hikari	5	F	
Japanese	12	Haruka	6	F	
Japanese	12	Ryo	7	M	
Japanese	12	Takeru	8	M	
Korean	13	Yumi	1	F	
Korean	13	Junwoo	2	M	
Korean	13	Hyeryun	4	F	
Korean	13	Jimin	5	F	
Korean	13	Sena	6	F	
Korean	13	Dayoung	7	F	
Korean	13	Hyuna	8	F	
Korean	13	Yura	9	F	
Korean	13	Jihun	10	M	
Thai	26	Sarawut	1	M	
Thai	26	Somsi	2	F	

Appendix B: Expressive Cues

NOTE: This feature is specific to Engine 2 voices.

Expressive Cues are a set of special tags which you may use in your text to specify distinct non-verbal expressions, such as laughing, crying, sighing, coughing, etc.

Expressive Cues tags are placed directly in your text.
For example, "clearing throat" sound:

_Throat_01 you may want to consider checking out our specials!

Or -

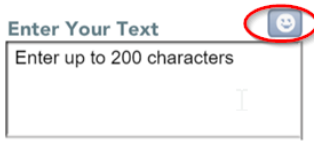
_Hurrah I am so happy to see you! _Whistle_01

NOTE: You must prepend all Expressive Cues with another '\' character when using them in API functions. For example:
sayText("Hello World _Laugh",5,1,2);

Every voice is unique in the Expressive Cues it supports. Some voices support many, others few. A complete list of Expressive Cue tags supported by each of the voices can be found in the Expressive Cues Listing.

You may want to try out the Expressive Cues in this convenient demo app.
Note: Demo app is implemented in Flash - but Flash is not required to use Expressive Cues.
http://www.sitepal.com/ttswidgetdemo

When you select a voice that supports Expressive Cues - the Expressive Cues button becomes active. Press it to select from the many Cues supported by that voice.



See the Expressive Cues Listing for detailed information.