Oddcast Text-To-Speech Module

Table of Contents

Overview

The Oddcast TTS Module (OTM) enables customers to use the Oddcast online TTS service, to generate & play text to speech audio in realtime within their online web pages, flash applications, or native code applications.
The OTM supports TTS in over 20 languages, with multiple voices available per each language. OTM provides access to these languages and voices via a client-side API.

OTM is a flash object, that a web page or application loads & can then access via API functions. The API supports the generation of synthetic audio from text and playback control functionality. Interfacing with OTM is entirely on the client side. OTM encapsulates interaction with Oddcast servers.

> Note: The straightforward way to interact with the generated audio, is to control it via the API functions described here. Using these high level functions you do not need to have direct access to the audio data. However, for those programmers who need low level access to the stream data, OTM allows direct access through the "audio_started" callback.

Introduction

To use these instructions you require 'embed code' unique to your Oddcast account.

The embed code will be provided by Oddcast support when your account is setup. The Embed code is specific to either Javascript or Flash-AS3 or Flash-AS2 - be sure to ask for what you need.

> *Note: C++ & C# are also supported, though this document does not include examples or instructions specific to those languages. Ask for more information.*

Your embed code is specific to your account and for your protection will allow playback only from your licensed domain(s). Please advise your Oddcast support person which domain(s) should be enabled for your account. Note: subdomains must be explicitly declared.

What you need to do:

1. Specify to your Oddcast support rep -
    a. Type of embed code you need - JS, AS2, AS3
    b. Domain(s) to activate.
2. Get embed code from Oddcast Support
3. Follow instructions and examples in this document to load the OTM and use the available API function calls.

Available resources:

- code eaxmples within this document
- source code examples to be provided by Oddcast Support.

Using your Embed Code

The embed code is a code segment provided by Oddcast to load the OTM into your web page or Flash application. Instructions and examples below explain how to incorporate the embed code.

JavaScript Instructions

Paste your embed code into your HTML page BODY section. 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.

Flash 9 AS3 Instructions

1. Run Macromedia 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 vh_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);
   vh_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:

```
vh_player.addEventListener(“vh_sceneLoaded”, vh_sceneLoaded);
```

Flash 8 AS2 Instructions

1. Run Macromedia 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 Macromedia Flash 8 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 vh_sceneLoaded(){
        //any commands that should be triggered here;
}
```

Playback Control Functions

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

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

Note: This function is available only to a TTS Enabled account and will work only within a licensed domain for the account. Domain specific licensing is necessary to avoid misappropriation of TTS streams. If the account is not 'TTS-Enabled', or playback is attempted within a domain that is not specifically licensed for the account, then this call will have no effect.

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.

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

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

`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

`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)
```

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

Preload a specific Text To Speech audio. Calling loadText in advance can reduce the loading time when the audio is played. Calling loadText a second time, while audio is loading or after audio has been loaded has no effect.
Implement the vh_ttsLoaded() event callback to be notified when the audio track is done loading. Use the sayText() function to play the audio.

Note: using loadText may be unnecessary. Use it only if your application requires split second timing coordination when starting audio playback.

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 B.

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

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

`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

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

Example:

```
loadText('Hello World',1,1,1)
loadText('Hello World',1,1,1,'D',3)
```

---

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 affects only the volume . To stop playback, use the function stopSpeech().

---

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

vh_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 "vh_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 can help improve coordination between OTM 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 in both cases, 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 2.0 (deprecated)
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 3.0
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 "vh_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("vh_talkStarted",talkStartedHa
ndler);
function talkStartedHandler():void{ trace("talk started"); }
}
```

vh_audioProgress (percentPlayed)

Called during playback, if and only if the 'progressInterval' status is set. vh_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 vh_audioProgress(percentPlayed) {
}
```

Example - ActionScript3

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

---

vh_talkStarted ()

Triggered when the audio playback begins.

Example - JavaScript & ActionScript2

```
function vh_talkStarted(){
}
```

Example - ActionScript3

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

---

vh_talkEnded ()

Triggered when audio playback is done.

Example - JavaScript & ActionScript2

```
function vh_talkEnded(){
}
```

Example - ActionScript3

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

---

vh_audioStarted ()

Triggered when audio playback begins. Unlike vh_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 vh_audioStarted(){
}
```

Example - ActionScript3

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

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

---

vh_audioEnded ()

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

Example - JavaScript & ActionScript2

```
function vh_audioEnded(){
}
```

Example - ActionScript3

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

---

vh_ttsLoaded (text)

Triggered when a Text-To-Speech audio preload is complete; returns the text that was provided as input to 'loadText'.

Arguments:

`text` preloaded audio text

Example - JavaScript & ActionScript2

```
function vh_ttsLoaded(audio_text){
sayText(audio_text,2,1,1);
}
```

Example - ActionScript3

```
MovieClip(loader.content).addEventListener("vh_ttsLoaded",ttsLoadedHandle
r);
function ttsLoadedHandler(event:*):void{
trace("text to speak: "+ event.data);
}
```

Appendix: Text to Speech Languages and Voices

The following tables list Voice Family IDs, Language IDs and Voice IDs available for use with the OTM.

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

TTS Voice Family: 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	√
Spanish	2	Carmen	1	F	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	Gabriela	1	F	Brazilian	√
Portuguese	6	Amalia	2	F	European	√
Portuguese	6	Eusebio	3	M	European	√
Portuguese	6	Fernanda	4	F	Brazilian	√
Portuguese	6	Felipe	5	M	Brazilian	√
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		√
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		√
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		√

* *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.

TTS Voice Family: 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
Spanish	2	Violeta	1	F	
Chinese	10	Lily	1	F	Mandarin
Chinese	10	Hui	3	F	Mandarin
Chinese	10	Liang	4	M	Mandarin
Japanese	12	Show	2	M	
Japanese	12	Misaki	3	F	
Korean	13	Yumi	1	F	
Korean	13	Junwoo	2	M	