Table of Contents

Introduction

The SitePal/Studio Player provides an API (Application Programming Interface) that allows a JavaScript or Flash ActionScript programmer to affect different runtime attributes by making function calls from the hosting HTML page or Flash movie. The API enables communication between the host environment and the embedded VHost Scene or Show.

API functions allow you to interface with Shows and Scenes embedded in Flash movies and HTML pages. Unless otherwise noted, the API functions are identical for both JavaScript and ActionScript environments. Where the usage or parameters differs, this is clearly noted.

> Note: SitePal users do not have access to Shows, only to Scenes. “Show specific” functions which apply only to Studio are clearly marked as such.

API functions work only after Scene or Show has completed loading

Certain aspects of the API may not function in a predictable manner until the “vh_sceneLoaded” status callback has been called/dispatched. It is therefore advisable to implement the “vh_sceneLoaded” callback – and not call any API functions until the “vh_sceneLoaded” callback has been invoked.

For more detailed information please review the Status Callback Functions section.

Embedding in an HTML page:

To use the functions in an HTML page you must add the ‘JavaScript API Stub Code’ to the HTML page in which the VHost object is embedded. The JavaScript API Stub Code is available in the ‘Embed’ window. Copy-paste the JavaScript stub code from the ‘Embed’ window into the HTML page.

When calling an API function from within a different frame or window, be sure to reference the page where the JavaScript API Stub Code is embedded.

Note: *Basic knowledge of JavaScript is assumed.*

Embedding in a Flash movie:

Embed a Show or a Scene in a Flash movie by using the ‘loadMovie’ ActionScript function. Once you load the VHost object into a movie instance, any call to a VHost API function should be formatted as: instance.function() , where “instance” is the name of the object into which the VHost object was loaded.

Note: Basic *knowledge of Flash ActionScript is assumed.*

Examples & Additional Reference Material

Before you get started – you may want to check out our comprehensive source code examples – please see reference links in Appendix A.

Function Reference

Animation Control Functions

followCursor(mode)

Available for: ☑Studio ☑SitePal

Turn “follow cursor” to the OFF, ON IN BOX, or ON IN PAGE state. If OFF, the character’s gaze remains fixed, and ignores cursor movement. If ON IN BOX, the character’s head and eyes follow the cursor within the embed rectangle. If ON IN PAGE, the character’s gaze follows the cursor in the entire page, including areas outside the flash embed rectangle.

Arguments:

`mode` Required, Numeric (0/1/2):

0: follow cursor is set to OFF.
1: follow cursor is set to ON IN BOX
2: follow cursor is set to ON IN PAGE.

Example:

```
followCursor(1)
```

---

freezeToggle ()

Available for: ☑Studio ☑SitePal

Toggle between the pause and play states. Pause – character falls asleep. If the character is speaking, speech is paused. Play - character wakes up. If the character was previously paused in mid-speech, speech resumes from that point.

Arguments:

None.

Example:

```
freezeToggle()
```

---

recenter()

Available for: ☑Studio ☑SitePal

Cause the VHost character to set its gaze to the default position.

Arguments:

None.

Example:

```
recenter()
```

setGaze(degrees, duration, [magnitude])

Available for: ☑Studio ☑SitePal

Set the direction & magnitude of the VHost character head and eye movement.

This call will cause the VHost character to divert the orientation of its gaze to the specified direction, and maintain the new orientation for the specified period of time. The orientation will naturally shift towards the center (default) position when the specified time is up, or when/if the VHost is requested to speak.

The optional *magnitude* parameter governs the "intensity" of the head & eye movement.

Arguments:

`degrees`	Required. Numeric. 0-360 (0 deg.=top, 90 deg.=right, etc.)
`duration`	Required. Numeric. In Seconds.
`magnitude`	Optional. Numeric. In percent. 0-100. Default = 100.

Example:

```
setGaze(90, 6)
```

---

setFacialExpression(expression, duration, amplitude)

Available for: ☑Studio ☑SitePal

Note: this call is only supported for 3D characters. If called for a 2D character, it has no effect. (See related function is3D).

Set the facial expression animation for a character. If "0" is passed no duration is required. setExpression calls do not queue, but interrupt. If a call is made while a previous call's duration is still in effect, the first expression transforms into the second expression immediately.

Arguments:

expression Required, Text string

"ClosedSmile" –	happy (closed mouth smile)
"OpenSmile" –	very happy (open mouth smile)
"Sad" –	sad
"Angry" –	angry
"Fear" –	afraid
"Disgust" –	disgusted
"Surprise"	surprised
"Thinking"	thinking
"Blush"	embarrassed (blush)
"LeftWink"	wink with left eye
"RightWink"	wink with right eye
"Blink"	blink with both eyes
"Scream"	mouth wide open for screem.

duration Required if expressionID!=0, Numeric.

Time in seconds. Use -1 for indefinite duration.

amplitude — Optional, Numeric. Range: 0, 100; default = 100
The extent to which the expression should be applied

Example:

```
setFacialExpression(3, 6);  // sets expression to sad for 6 seconds at max amplitude
```

---

clearExpressionList( )

Available for: ☑Studio ☑SitePal

Note: This function is only supported only for 3D characters. If called for 2D character it will have no effect.

Clear all expressions.

Example:

```
clearExpressionList();
```

---

setIdleMovement(frequency, amplitude)

Available for: ☑Studio ☑SitePal

Note: This function is fully supported only for 3D characters. If called for a 2D character, frequency is interpreted as follows: 0 – turn OFF idle movement; non-zero – turn ON idle movement. Radius is ignored.

Characters that are not engaged in speaking, following the mouse, or gazing (via the *setGaze* api) will randomly look around by default. This function enables users to set the frequency and intensity of the character's movement when not otherwise engaged.

Arguments:

`frequency` — Optional. Numeric. The frequency with which the character performs idle time head movement. Values are 0 to 100. Default is 50.
Use 0 to turn off Idle movement.

`amplitude` — Optional. Numeric. The distance from center the character sets its random gaze. Values are 1 to 100. Default is 50.

Example:

```
setIdleMovement(20,100);
```

---

setSpeechMovement(amplitude)

Available for: ☑Studio ☑SitePal

Note: This function is only supported for 3D characters.

Characters perform random head movements during speech. This function enables users to set the intensity of the character's movement when speaking or disable the movement altogether.

Arguments:

`amplitude`	Numeric. The intensity with which the character performs head movements while speaking. Values are 0 to 100. Default is 50.

Example:

```
setSpeechMovement(100);
```

---

animate(animationName)

Available for: ☑Studio ☑SitePal

Note: This function is only supported for FullBody characters.
The character can be directed to perform a specific animation. The list of animation names available for use will be made available at a later time. For the time being this function is available for internal use only.
When function is called, behavior is subject to the status of the "interrupt mode" status. Use 'setStatus' to update the interrupt mode.

- If the character is speaking, and interrupt mode is on, speech is interrupted and the character animates.
- If the character is speaking, and interrupt mode is off, no action is taken.
- If the character is not speaking, and interrupt mode is on, idle animation is interrupted and the character animates.
- If the character is not speaking, and interrupt mode is off, the character animates after current idle animation concludes and returns to idle state.

Arguments:

`animationName`	The name of the animation to be invoked.

Example:

```
animate('wave');
```

Speech Functions

loadAudio(name)

Available for: ☑Studio ☑SitePal

Preload a specific audio track by name. Calling loadAudio in advance can reduce the loading time when the audio is played. Calling loadAudio a second time, while audio is loading or after audio has been loaded has no effect.

Implement the vh_audioLoaded() event callback to be notified when the audio track is done loading.

Use the sayAudio() function to play the audio.

Arguments:

`name`	Required. String. The name of the audio track from the account.

Example:

```
loadAudio('audioname')
```

---

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

Available for: ☑Studio ☑SitePal

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.

Arguments:

`txt` Required. String - The text to speak. Text is limited to 900 characters. (225 characters in Chinese & Japanese). Longer text 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 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
- "W" – Whisper levels: 1, 2, 3

`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)
```

---

sayAudio(name, [startTime])

Available for: ☑Studio ☑SitePal

Play a specific audio track by name.

Arguments:

`AudioTrackName`	Required. String. The logical name of the audio as specified within the account.
`startTime`	Optional. Floating. The offset, in seconds, from the beginning of the audio from which to start audio playback.

Example:

```
sayAudio('audio name',1.9)
```

---

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

Available for: ☑Studio ☑SitePal

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

For detailed step by step instructions, please review the Guidelines for Using the TTS API.

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 a security measure. If the account is not TTS enabled, or the Scene is used within a domain that you have not included within your licensed domains, then this call will generate an alert. To edit your licensed domains please login to your account and select Account Info.

Arguments:

`txt`	Required. String - The text to speak. Text is limited to 900 characters. (225 characters in Chinese & Japanese). Longer text 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 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
	• "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)
```

---

sayAIResponse (txt,voice,lang,engine,[botID],[effect], [effLevel])

Available for: ☑Studio ☑SitePal

An Artificial Intelligence knowledge base provides a real time text and audio response to a text "question". The Audio is generated & spoken automatically via a Text To Speech engine according to the selected voice, language and engine. The response text is provided via the Event function 'vh_aiResponse()'.

The Artificial Intelligence knowledge base is based on the extensive A.L.I.C.E. knowledge base, which includes over 23,000 data entries. Your knowledge base can be edited and customized in the AI Management Center (AIMC) – click on AIMC from the main menu within your account.

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 a security measure. If the account is not TTS enabled, or the Scene is used within a domain that you have not included within your licensed domains, then this call will generate an alert. To edit your licensed domains please login to your account and select Account Info.

Arguments:

`txt` Required. String - The text to speak. Text is limited to 900 characters. (225 characters in Chinese & Japanese). Longer text 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 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
- "W" – Whisper levels: 1, 2, 3

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

Example:

```
sayAIResponse('Sing me a song',2,1,2)
sayAIResponse('Sing me a song',2,1,2,,'P',-1)
```

---

saySilent (seconds)

Available for: ☑Studio ☑SitePal

Speech is visually simulated: No audio is downloaded, no streams are consumed and no interaction with the server is performed.

This function call may be useful when you want to call attention to the character but do not want use audio to do so. Most pertinent example is use in ad banners, where (in some cases) the use of audio may only be permitted after mouse rollover.

saySilent is always in 'InterruptMode' ON, meaning that any function call which invokes actual speech will interrupt simulated speech. saySilent calls cannot be queued.

Arguments:

`Seconds` length of time desired for simulated speech, in seconds

Example:

```
saySilent(10)
```

---

setPlayerVolume (level)

Available for: ☑Studio ☑SitePal

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 the speech (lip movement continues) or stop the audio stream. It affects only the volume . To stop the speech, use the function stopSpeech().

---

stopSpeech ()

Available for: ☑Studio ☑SitePal

Stop the speech of a currently speaking VHost character. If the character is not currently speaking, stopSpeech has no effect (i.e. it does not prevent speech that has not yet begun).

Arguments:

None.

Example:

```
stopSpeech()
```

---

replay (ignorelimit)

Available for: ☑Studio ☑SitePal

Plays or replays current Scene, from the start.

If `interruptMode` is ON, ongoing playback (if any) is interrupted, and immediately begins again. If `interruptMode` is OFF, playback is queued. See setStatus to learn about `interruptMode`.

Playback limit is a Scene attribute which can be set in the Scene Options dialog. Set the `ignoreLimit` parameter to override this Scene option.

Arguments:

`ignoreLimit`	Optional. Boolean default value is 0. If 1 then Scene ignores playback limit status.

Example:

```
replay()
replay(1)
```

Speech Functions for Exported Scenes

sayAudioExported(name, accountId, [startTime])

Available for: ☑Studio ☐SitePal

Note: This call is only supported for Studio customers whose account has been configured to work with exported Scenes. Contact support@oddcast.com for more information.

Play a specific audio track by name, from an exported Scene.

Arguments:

`AudioTrackName`	Required. String. The logical name of the audio as specified within the account.
`accountId`	Verified VHSS account id
`startTime`	Optional. Floating. The offset, in seconds, from the beginning of the audio from which to start audio playback.

Example:

```
sayAudioExported('audio name', xxxx, 1.9)
```

---

sayTextExported(txt,voice,lang,engine,accountId,[effect], [effLevel])

Available for: ☑Studio ☐SitePal

Note: This call is only supported for Studio customers whose account has been configured to work with exported Scenes. Contact support@oddcast.com for more information.

Real-time (dynamic) Text-To-Speech (TTS), called from an exported Scene.

For detailed step by step instructions, please review the Guidelines for Using the TTS API.

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 a security measure. If the account is not TTS enabled, or the Scene is used within a domain that you have not included within your licensed domains, then this call will generate an alert. To edit your licensed domains please login to your account and select Account Info.

Arguments:

`txt`	Required. String - The text to speak. Text is limited to 900 characters. (225 characters in Chinese & Japanese). Longer text 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 in Appendix B.
`accountId`	verified VHSS account id
`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:

```
sayTextExported('Hello World',1,1,1,xxxx)
sayTextExported('Hello World',1,1,1,xxxx,'S',-2)
```

---

sayAIResponseExported (txt, voice, lang, engine, accountId, [botID],[effect], [effLevel])

Available for: ☑Studio ☐SitePal

Note: This call is only supported for Studio customers whose account has been configured to work with exported Scenes. Contact support@oddcast.com for more information.

An Artificial Intelligence knowledge base provides a real time text and audio response to a text "question". The Audio is generated & spoken automatically via a Text To Speech engine according to the selected voice, language and engine. The response text is provided via the Event function 'vh_aiResponse()'.

The Artificial Intelligence knowledge base is based on the extensive A.L.I.C.E. knowledge base, which includes over 23,000 data entries. Your knowledge base can be edited and customized in the AI Management Center (AIMC) – click on AIMC from the main menu within your account.

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 a security measure. If the account is not TTS enabled, or the Scene is used within a domain that you have not included within your licensed domains, then this call will generate an alert. To edit your licensed domains please login to your account and select Account Info.

Arguments:

`txt` Required. String - The text to speak. Text is limited to 900 characters. (225 characters in Chinese & Japanese). Longer text 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 in Appendix B.

`accountId` Verified VHSS account id

`botID` Optional. String – The ID of the AI bot to use for generating the response. This parameter is only relevant for Studio accounts which support multiple bots per account. SitePal users do not need to provide this parameter.

`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.

Example:

```
sayAIResponseExported('Sing me a song',2,1,2,xxxx)
sayAIResponseExported('Sing me a song',2,1,2,xxxx,,'P',-1)
```

Scene Attributes

setBackground (bgName)

Available for: ☑Studio ☑SitePal

Background is modified for playback session in progress. Change is not persistent.

Arguments:

`bgName` Required. String. The logical name of the background as specified within the account.

Example:

```
setBackground('background name')
```

---

setColor (part,color)

Available for: ☑Studio ☑SitePal

Dynamically change the color of the part specified. Colors are based on a gray offset in order to maintain the shading and detail. Therefore, the effect of a color on the specified area may not exactly match the input hex color value. The results may differ slightly from one character to another.

Arguments:

`part` Required. String. The character part to color.

One of: ‘eyes’, ‘hair’, ‘make-up’, ‘mouth’, ‘skin’

`color`. Required. String. Hexadecimal RGB color representation.

Example:

```
setColor('eyes','0000AA')
```

---

setLink (href,target)

Available for: ☑Studio ☑SitePal

Set the URL address for the link of the currently executing scene. For this call to have an effect, the scene should have a specified Trigger to activate the link.

Arguments:

`href` Required. String. The URL address for the link.

`target` Optional. String . A frame name or a window name

An optional parameter specifying the window or HTML frame that the document should load into. For ‘Window’ you can enter the name of a specific window or choose from the following reserved target names:

_self	specifies the current frame in the current window
_blank	specifies a new window
_parent	specifies the parent of the current frame
_top	specifies the top-level frame in the current window

If no value is specified, default value is “_blank”.

Example:

```
setLink('http://www.yoursite.com','_blank')
```

---

setStatus (interruptMode,progressInterval,gazeSpeed)

Available for: ☑Studio ☑SitePal

This function is used to 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 and sayAudio) are queued for consecutive playback.

If set to 1 current audio is interrupted when sayAudio or sayText are 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.

`gazeSpeed`

Required. Integer (0/1/2) – Default = 0.

controls the reaction speed of the character when responding to setGaze function calls.

0 - slow
1 - medium
2 - fast

Example:

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

---

dynamicResize (width, height)

Available for: ☑Studio ☑SitePal

The dimensions of the embedded Show or Scene are dynamically modified without reloading the character. This can be used to support responsive design. Change is not persistent - if the page is reloaded, the embedded Show or Scene will load as originally embedded.

We recommend that original aspect ratio be maintained when the Scene is resized. If you want to maintain the same relative position of the character within the Scene frame, you should retain the same aspect ratio. Otherwise, character will be re-positioned as best possible.

Note: To control the dimensions of the embedded Show or Scene when page is initially loaded, you could rewrite the width and height embed values *before* the page loads, using a back end programming language such as Java or php.

```
<scripttype="text/javascript">AC_VHost_Embed(accountid,height,width,'',1,1
, showsceneid, 0,1,0,'94fb29b9a4767343f36dd16fc8c0f81a',0);</script>
```

Arguments:

`width`	Required. Integer. The new width.
`height`	Required. Integer. The new height.

Example:

```
resize(300,200)
```

---

is3D ()

Available for: ☑Studio ☑SitePal

Is the character in the current Scene a 3D character?

Boolean function – returns true if 3D character is used, false otherwise.

Arguments:

None.

Example:

```
is3D()
```

Embed Overlay Functions

The following functions apply only to Scenes and Shows that are embedded as an overlay on top of the page. If any of these functions is called for a Scene or Show which is embedded IN the page, the function call will have no effect.
Note: Embed overlay functions are available in JavaScript but not in ActionScript.

overlayOpen (mode, play)

Available for: ☑Studio ☑SitePal

Scene or Show is opened, or toggled between minimize and maximize display mode. If mode is 'max' then play parameter governs the playback behavior of the Scene/Show when opened.

Arguments:

`mode` 'max' - open Scene/Show in maximized mode or toggle to maximize mode. If the Scene is already maximized, this call has no effect.
'min' - open Scene/Show in minimized mode or toggle to minimize mode. The effect is equivalent to a click on the minimize button. If the Scene is already minimized, this call has no effect.

`play` Optional. Integer (0/1/2) – Default = 2.
Relevant only for 'max' mode. If mode is 'min' then parameter is ignored.
If set to 0, playback does not start. Settings ignored
If set to 1, playback immediately starts. Settings ignored.
If set to 2, playback may start depending on the values of the "playback limit" and "play on load" settings.

Examples:

```
overlayOpen('max',1)
overlayOpen('max')
overlayOpen('min')
```

---

overlayClose ()

Available for: ☑Studio ☑SitePal

Scene or Show is closed. The effect is equivalent to a click on the close button. If Scene is already closed, this call has no effect.

Arguments:

None.

Example:

```
overlayClose()
```

Navigation Flow Functions

gotoNextScene ()

Available for: ☑Studio ☐SitePal

The current Scene is interrupted, and the next Scene (according to the preset show flow) immediately begins. This has the same effect as pressing the player's 'Next' button.

Arguments:

None

Example:

```
gotoNextScene()
```

---

gotoPrevScene ()

Available for: ☑Studio ☐SitePal

The current Scene is interrupted, and the previous Scene immediately begins. This has the same effect as pressing the player's 'Previous' button.

Arguments:

None

Example:

```
gotoPrevScene()
```

---

gotoScene (sceneRange)

Available for: ☑Studio ☐SitePal

The current Scene is interrupted, and the specified scene immediately begins.

Arguments:

`sceneRange`	Required. String Indicates the next Scene to follow the currently playing Scene. Can be either: - Index of specific Scene - Range of consecutive scenes to randomly choose from. The format is hyphen delimited.

Example

```
gotoScene('3')        – goto a specific Scene
gotoScene('4-7')      – goto a randomly selected scene from the set: 4,5,6,7
```

---

loadScene (sceneIndex)

Available for: ☐Studio ☑SitePal

SitePal ONLY - The current Scene is interrupted, and the specified Scene is loaded instead.

Arguments:

`sceneIndex`	Required. Integer Indicates the Scene to load.

Example

`loadScene(3)` – load Scene #3

---

loadShow (showIndex)

Available for: ☑Studio ☐SitePal

Studio ONLY - The current Show is interrupted, and the specified Show is loaded instead.

Arguments:

`showIndex`	Required. Integer Indicates the Show to load.

Example

`loadShow(3)` – load Show #3

---

preloadNextScene ()

Available for: ☑Studio ☐SitePal

Preloads the assets of the next scene in a show. Upon successful preloading of a scene the callback vh_scenePreloaded() will be invoked. If there is no next scene the call is ignored. Subsequent calls to this function or preloadScene while a scene is loading will be ignored.

Arguments:

None

Example:

```
preloadNextScene()
```

---

preloadScene (sceneIndex)

Available for: ☑Studio ☐SitePal

Preloads the assets of the specified scene. Upon successful preloading of a scene the callback vh_scenePreloaded() will be invoked. If there is no scene with the specified index number the call is ignored. Subsequent calls to this function or preloadNextScene while a scene is loading will be ignored.

Arguments:

`sceneNumber`	Required. Integer The index of the scene to preload

Example

`preloadScene(3)` – preload a specific Scene

setNextSceneIndex (sceneRange)

Available for: ☑Studio ☐SitePal

Set the next Scene in the playback order. Actual transition to the specified Scene will not take place until a user clicks on the player's Next button or the function gotoNextScene () is called. Setting the sceneIndex is equivalent of setting the 'Next Scene' attribute in the VHost Show Editor.

Arguments:

`sceneRange`	Required. String. Indicates the next scene to follow the currently playing scene. Can be either: - Index of specific scene - Range of scenes that specify a set of scene indexes. In this case one of the specified set is selected at random. The format is comma delimited, where consecutive ranges can be specified by a hyphen.

Examples:

`setNextSceneIndex('3')`	– the next scene is a specific scene
`setNextSceneIndex('1,4-6,12')`	– next scene is a random selection from the set: 1,4,5,6,12

Status Callback Functions

Status Callback Functions (SCFs) can be useful when embedding a Scene or Show in your web page or Flash application. Using SCFs can help improve coordination between the embedded Scene or Show and your page or application.

SCFs 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 (depracated)

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 SCF. 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 Scene's flash embed code here */ );
function setListeners():void
{
MovieClip(loader.content).addEventListener("vh_talkStarted",talkStartedHandler);
function talkStartedHandler():void{ trace("talk started"); }
}
```

API functions work only after Scene or Show has completed loading

Keep in mind that certain aspects of the API may not function in a predictable manner until the "vh_sceneLoaded" status callback has been called/dispatched. It is therefore advisable to always implement the "vh_sceneLoaded" callback.

vh_aiResponse ()

Available for: ☑Studio ☑SitePal

Triggered when an AI Response is returned, this call returns the text that is generated by the AI knowledge base in response to the function call 'sayAIResponse'.

Arguments:

`response_text`	Response text

Example - JavaScript & ActionScript2

```
function vh_aiResponse(response_text){
}
```

Example - ActionScript3

```
MovieClip(loader.content).addEventListener("vh_aiResponse",aiResponseHandler);
function aiResponseHandler(event:*):void{
	trace("ai response text: "+ event.data);
}
```

vh_audioLoaded (audioName)

Available for: ☑Studio ☑SitePal

Triggered when an audio preload is done, and returns the name of the audio that was provided as input to 'loadAudio()'.

Arguments:

`audio_name` Loaded audio name

Example - JavaScript & ActionScript2

```
function vh_audioLoaded(audio_name){
      sayAudio(audio_name);
}
```

Example - ActionScript3

```
MovieClip(loader.content).addEventListener("vh_audioLoaded",audioLoadedHandler);
function audioLoadedHandler(event:*):void{
      trace("audio name: "+ event.data);
}
```

vh_ttsLoaded (text)

Available for: ☑Studio ☑SitePal

Triggered when a Text-To-Speech audio preload is done and returns the text that was provided as input to 'loadText()'.

Arguments:

`audio_text` Loaded audio text

Example - JavaScript & ActionScript2

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

Example - ActionScript3

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

vh_audioProgress (percentPlayed)

Available for: ☑Studio ☑SitePal

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 status.

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",audioProgHandler);
function audioProgHandler(event:*):void{
	trace("percent played: "+ event.data.percent);
}
```

---

vh_sceneLoaded (sceneIndex)

Available for: ☑Studio ☑SitePal

Triggered when the Scene is fully loaded & displayed, just before the audio starts playing. Use this callback to verify Scene is ready to accept API calls.

Note: sceneIndex parameter is only relevant to Studio accounts.

Arguments:

`sceneIndex`	For Studio accounts: The index of the loaded Scene in the Show. Undefined for SitePal accounts.

Example - JavaScript & ActionScript2

```
function vh_sceneLoaded(sceneIndex){
	alert(''the scene has started'');
}
```

Example - ActionScript3

```
MovieClip(loader.content).addEventListener("vh_sceneLoaded",sceneLoadedHandler);
function sceneLoadedHandler(event:*):void{
	trace("scene started. index: "+ event.data);
}
```

---

vh_scenePreloaded (sceneIndex)

Available for: ☑Studio ☐SitePal

Triggered after a successful call to preloadScene or preloadNextScene. The assets of the scene are loaded to memory. Subsequent display of the specified scene should be immediate.

Arguments:

`sceneIndex`	For Studio accounts: The index of the loaded Scene in the Show.

Example - JavaScript & ActionScript2

```
function vh_scenePreloaded(sceneIndex){
	alert(''the scene is preloaded. index: "+ sceneIndex);
}
```

Example - ActionScript3

```
MovieClip(loader.content).addEventListener("vh_scenePreLoaded",pLoadedHandler);
function pLoadedHandler(event:*):void{
	trace("scene preloaded. index: "+ event.data);
}
```

---

vh_talkStarted ()

Available for: ☑Studio ☑SitePal

Triggered when the VHost character starts talking

Arguments

`caller`	Path to caller (Flash only). Not in use.

Example - JavaScript & ActionScript2

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

Example - ActionScript3

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

---

vh_talkEnded ()

Available for: ☑Studio ☑SitePal

Triggered when the VHost character is done talking

Arguments

`caller`	Path to caller (Flash only). Not in use.

Example - JavaScript & ActionScript2

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

Example - ActionScript3

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

---

vh_audioStarted ()

Available for: ☑Studio ☑SitePal

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",audioStarted
Handler);

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

---

vh_audioEnded ()

Available for: ☑Studio ☑SitePal

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

Example - JavaScript & ActionScript2

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

Example - ActionScript3

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

---

vh_playPause (status)

Available for: ☑Studio ☑SitePal

Triggered when the play/pause button is pressed. This enables synchronization

Arguments:

`status`	0=paused; 1=playing.

Example - JavaScript

```
function vh_playPause(status){
      alert(''play/pause button pressed. status: "+ status);
}
```

Appendix A: API Examples

For full source code examples of using the Client API please see:

Using the Client API in Javascript
Using the Client API in ActionScript 3 (Flash 9)
Using the Client API in ActionScript 2 (Flash 8)
Advanced Client API examples

Additional reference material can be found in our support section.

Appendix B: Text to Speech Languages and Voices

The following tables list all Voice Family IDs, Language IDs and Voice IDs supported by the sayText & sayAIResponse API calls.

Language	*ID*
Arabic	27
Basque	22
Catalan	5
Chinese	10
Czech	18
Danish	19
Dutch	11
English	1
Esperanto	31
Finnish	23
French	4
Galician	15
German	3
Greek	8
Hindi	24
Hungarian	29
Indonesian	28
Italian	7
Japanese	12
Korean	13
Norwegian	20
Polish	14
Portuguese	6
Romanian	30
Russian	21
Spanish	2
Swedish	9
Thai	26
Turkish	16

TTS 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 Loquendo voices, as indicated above. For a complete list of Expressive Cue tags see Appendix D.

TTS 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	European
French	4	Louis	4	M	European
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	Hayuna	8	F	
Korean	13	Yura	9	F	
Korean	13	Jihun	10	M	
Thai	26	Sarawut	1	M	
Thai	26	Somsi	2	F	

TTS Engine ID = 4

Language	Lang. ID	Voice Name	Voice ID	Gender	Description
English	1	Jill	2	F	US
English	1	Tom	3	M	US
English	1	Karen	4	F	Australian
English	1	Daniel	5	M	UK
English	1	Serena	7	F	UK
English	1	Moira	8	F	Irish
English	1	Sangeeta	9	F	Indian
English	1	Lee	10	M	Australian
English	1	Samantha	11	F	US
English	1	Fiona	12	F	Scottish
English	1	Tessa	13	F	South African
Spanish	2	Duardo	1	M	
Spanish	2	Monica	3	F	
Spanish	2	Paulina	4	F	Mexican
Spanish	2	Javier	5	M	Mexican
German	3	Steffi	1	F	
German	3	Yannick	2	M	
German	3	Anna	3	F	
French	4	Felix	1	M	Canadian
French	4	Julie	2	F	Canadian
French	4	Sebastien	3	M	European
French	4	Virginie	4	F	European
French	4	Thomas	5	M	European
Catalan	5	Nuria	1	F	
Portuguese	6	Raquel	2	F	Brazilian
Portuguese	6	Joana	3	F	European
Italian	7	Paolo	1	M	
Italian	7	Silvia	2	F	
Greek	8	Alexandros	1	M	
Swedish	9	Alva	1	M	
Swedish	9	Oskar	3	M	
Chinese	10	Sin-Ji	1	F	Cantonese
Chinese	10	Ya-Ling	2	F	Taiwanese Mandarin
Chinese	10	Ting-Ting	4	F	Mandarin
Dutch	11	Ellen	1	F	Belgian
Dutch	11	Clair	2	F	
Dutch	11	Laura	3	F	
Dutch	11	Xander	4	M	
Japanese	12	Kyoko	1	F	

Korean	13	Narae	1	F
Polish	14	Agata	1	F
Turkish	16	Aylin	1	F
Czech	18	Zuzana	1	F
Danish	19	Ida	1	F
Norwegian	20	Stine	2	F
Russian	21	Milena	2	F
Basque	22	Arantxa	1	F
Finnish	23	Mikko	1	M
Hindi	24	Lekha	1	F
Thai	26	Narisa	1	F
Arabic	27	Maged	1	M
Indonesian	28	Damayanti	1	F
Hungarian	29	Eszter	1	F
Romanian	30	Simona	1	F

Appendix C: SSML Tags for Text to Speech

The Speech Synthesis Markup Language (SSML) is an XML based language used to represent instructions to Text-To-Speech engines when processing input text. SSML Tags are inserted within the actual text to be processed, and are subsequently interpreted by the TTS engine to affect the manner in which voice audio is generated.

Using SSML Tags in your input text is not necessary, but allows you to achieve more precise control over the manner in which the text is spoken.

The syntaxt for SSML is an emerging standard, governed by the W3C. The specification for SSML 1.0 has only recently been finalized (see SSML Specification for more information). It should therefore come as no surprise that support for SSML is not yet fully or uniformly implemented.

We have reviewed what we consider the most relevant tags, and verified their implementation and functionality within the available TTS Engines. The following list summarizes our findings. For each of the listed tags, we note the support status per each of the TTS Engines (a.k.a Voice Family) #2 and #3 (Loquendo and Neospeech). Note that where specific languages are mentioned, this means that other languages for that TTS Engine have been reviewed and are not supported. This list will be updated from time to time.

Note: TTS Engine #4 does not support SSML tags. Please select only voices from Engines #2 and #3 for use with SSML.

Additional SSML tags, which are part of the SSML Specification but not listed here, might be useful for your purposes. Please feel free to experiment and come to your own conclusion regarding the suitability of unlisted tags.

Note that SSML tag interpretation is case sensitive, and the case of opening and closing tags must match!

Examples:

```
<Prosody volume="loud"> very loud </prosody> Wrong
<prosody volume="loud"> very loud </prosody> Correct
<Prosody volume="loud"> very loud </Prosody> Correct
```

Structure Elements

Break

The **Break** tag instructs the TTS engine to insert a pause in the synthesized text in one of three ways.

Loquendo: **Partial support.**

Loquendo does not support the "size" attribute of the <break /> element, only the "time" attribute.

Neospeech: **Supported**

Syntax: `<BREAK/>`

Example: `Time for a pause <Break/> Okay, keep going.`

Inserts a brief break after the word "pause".

Syntax: `<BREAK Size="none | small | medium | large"/>`

Example: `No time for a pause <Break size="none"/> Keep going.`

Inserts no break after the word "pause".

Example: `Time for a pause <Break size="medium"/> Okay, keep going.`

Inserts a brief silence, the equivalent of the silence following a sentence, after the word "pause".

Example: `Time for a pause <Break size="large"/> Keep going.`

Inserts only the default break after the word "pause".

Example: `Time for a pause <Break size="medium"/> Okay, keep going.`

Inserts the equivalent of a paragraph break of silence after the word "pause".

Syntax: `<BREAK time=" duration "/>`

Example: `Break for 100 milliseconds <Break time="100ms"/> Okay, keep going.`

Inserts 100 milliseconds of silence after the word "milliseconds".

Example: `Break for 3 seconds <Break time="3s"/> Okay, keep going.`

Inserts 3 seconds of silence after the word "seconds".

Paragraph

The **PARAGRAPH** tag tells the TTS engine to change the prosody to reflect the end of a paragraph, regardless of the surrounding punctuation.

Syntax: `<PARAGRAPH> text </PARAGRAPH>`

`<P> text </P>`

Loquendo: **Supported**

Neospeech: **Supported**

Example: `<Paragraph> This example has only one sentence in the paragraph </Paragraph>`

Example: **<**`P`**>** `The paragraph tag can be abbreviated as just the letter P.` **<**`/P`**>**

The TTS engine changes the prosody to reflect the paragraph boundaries.

Sentence

The **SENTENCE** tag tells the TTS engine to change the prosody to reflect the end of a sentence, regardless of the surrounding punctuation.

Syntax: `<SENTENCE> text </SENTENCE>`

`<S> text </S>`

Loquendo: **Supported**

Neospeech: **Supported**

Example: `<Sentence> This text is a sentence. </Sentence>`

Example: `<S> The sentence tag can be abbreviated as just the letter S. </S>`

The TTS engine changes the prosody to reflect the sentence boundaries.

Prosody Elements

Volume

The **Volume** attribute of the **Prosody** tag allows the application to change the volume of the TTS voice. Note that this does not change the volume of the output device, but it does raise or lower the volume of the text spoken within the context of the tag.

Syntax: `<PROSODY VOLUME=" level "> text </PROSODY>`

where *level* is a value from 0.0 to 200.0. A value of 100 is the voice's default volume, a value of 0 changes the volume to 0 and a value of 200 doubles the volume. The volume changes linearly.

Syntax: `<PROSODY VOLUME=" silent | soft | medium | loud "> text </PROSODY>`

Sets the absolute volume to the specified level.

Loquendo: **Supported**

Neospeech: **Supported**

Example:
```
This is the default volume
<prosody volume="silent"> silence </prosody>
<prosody volume="soft"> Now I'm whispering </prosody>
<prosody volume="120"> a little louder </prosody>
<prosody volume="medium"> medium volume</prosody>
<prosody volume="loud"> very loud </prosody>
```

Rate

The **RATE** attribute of the **Prosody** tag changes the rate at which the text is spoken. You can specify either the absolute rate or a relative change in the current speaking rate.

Syntax: `<PROSODY RATE="x-fast | fast | medium | slow | x-slow | default"> text </PROSODY>`

Syntax: **`<PROSODY RATE="`***`relativeChange`***`"> `***`text`*** **`</PROSODY>`**

changes the speaking rate which is expressed in Words Per Minute (WPM) or in percentage terms. *relativeChange* is a floating point number that is added to or subtracted from the current rate. A "+" or "-" sign must precede the number. If a percent sign follows then the change is interpreted as a percentage change..

Loquendo: **Supported**

Neospeech: **Supported**

Example:
```
This is the default speed
<prosody rate="slow"> this is speaking slowly
   <prosody rate="fast"> this is speaking fast </prosody>
   back to slow
</prosody>
back to the default rate
```

Example:
```
This is the default speed
<prosody rate="-50%">
    this is 50% slower
    <prosody rate="+50%"> this is 50% faster </prosody>
   back to 50% slower
</prosody>
back to the default rate
```

Pitch

The **PITCH** attribute of the **Prosody** tag changes the pitch at which the text is spoken. You can specify either the absolute pitch or a relative change in the current speaking

pitch.

Syntax: `<PROSODY PITCH=”x-high | high | medium | low | x-low | default”> text </PROSODY>`

Syntax: `<PROSODY PITCH=”relativeChange”> text </PROSODY>`

relativeChange is an floating point number, expressed as a percentage that is added to or subtracted from to the current pitch. A “+” or “-“ sign must precede the number, and the percent sign must follow.

Loquendo: **Supported**

Neospeech: **Supported**

Example: `<prosody pitch="+12.5%"> Higher pitch sentence </prosody>`
Example: `<prosody pitch="high"> High pitch sentence </prosody>`

The Voice Element

The **Voice** tag enables control the voice of the TTS speaker from the input text. You can use this feature to change voices, e.g. you might use different voices to speak different sections of an email message or carry on a conversation between two different voices. You can even use different languages within the same sentence.

Note: this can only work when switching voices within the same voice family.

Select a voice by specifying one of the following attributes:

Gender, **Name**.

It is best to specify the speaker by **Name**, in which case the Gender attribute is unnecessary.

Syntax:
```
<VOICE
    Gender="male | female | neutral"
    Name= voicename
</VOICE>
```

Loquendo: **Supported by some voices**

Neospeech: **Supported**

Example:
```
<voice name="Bridget"> This is Bridget, <Voice
Name="Violeta"> Hola, me llamo Violeta,</Voice> and this
is Bridget again. </voice>
```

This string is pronounced in Bridget's voice "This is Bridget", then in Violeta’s voice in Spanish, "Hola, me llamo Violeta", then in Bridget’s voice, "This is Bridget again".

Appendix D: Loquendo Expressive Cues

NOTE: This feature is specific to Loquendo (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

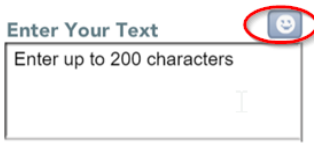
NOTE: You must prepend all Expressive Cues with another '\' character when using them in API functions. For example:

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 can try out the Expressive Cues in this demo -
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.