SitePal Client API Reference

The SitePal Client API is a collection of Javascript functions that enable full programmatic control over SitePal avatars in your web pages & integration with your business logic.

Table of Contents

Introduction

The SitePal avatar playback environment supports an API that allows you to control character speech and other runtime attributes by making JavaScript function calls from your web page, and receiving status 'callbacks'. The API enables communication between your web page and the embedded Scene.

Note: the terms 'character' and 'avatar' are used interchangeably in all SitePal documentation. They mean the same thing.

Embedding SitePal in your web page

To use this API you must first add your embed code to your web page. In your SitePal account, click on the "Publish" button for the Scene you wish to embed. Select the "Embed" option, copy your embed code and add it to the BODY section of your page, where you wish your character to appear.

Note: This API is fully supported in JS frameworks, but the embed mechanism is different. If you are using a JS framework (such as Angular, React, Vue, NextJS, React Native, and others that we support) see special instructions & examples on our support page.

Important caveats / pitfalls to avoid -

- API function calls will not work as expected until your embedded Scene is fully loaded. It is therefore advisable to implement the 'vh_sceneLoaded' callback – and not call any API function before this callback is received. For more information please review the Callback Functions section below.
  For example, if you want to lower the blink rate (of the eyes), use this code:

```
function vh_sceneLoaded(){
        setBlinking(20); // Set blink rate to 20/100 (0 to disable)
}
```

- For your protection certain API functions only work when your page is loaded from a domain you authorize. Such domains are called *Licensed Domains,* Specifically:
  - 'sayText' and 'sayAI' only work when your page is loaded from a Licensed Domain.
  - If you turn ON 'Secure Playback' for your account, this restriction will be applied to all functionality & speech. Note: default state is OFF.
    If you see an alert message when your SitePal loads or attempts to speak – then you may need to setup the Licensed Domain for your web page.
  - Add/edit your Licensed Domains in your *Settings* page.
  - Wildcards are supported in the domain name's first segment. So for example, you may specify `*.mycompany.com` to cover all subdomains.
  - 'localhost' and '127.0.0.1' are always authorized and do not need to be specifically declared.

Examples & Additional Reference Material

As you further review this documentation, it may be helpful to check out our comprehensive technical examples which cover the use of all functions in this API, as well as multiple advanced scenarios. See reference links in Appendix A.

Note: recently added features, functions or details are highlited in yellow for your convenience.

Play on Load (Autoplay)

On web browsers audio playback in a web page must be preceded by user interaction with the page (e.g. user clicks on a button, or touches the screen). This user action "activates" the web page which will then allow audio playback. This restriction is intended to prevent a web page from playing audio unprompted.

Attempting to do so using this API will not cause a problem – but may not work, depending on the browser & the circumstances.

On desktop browsers, the restriction is not absolute. Some browsers will allow play-on-load for a user who has visited your page before and interacted with media on the page. The policies enacted by browsers in this regard are both evolving & undocumented.

The main takeaways therefore should be:

- It is ok to try to initiate speech as soon as the page loads (verify that API has loaded first! - see 'caveats' above).
- You should be aware that such play-on-load attempts may be blocked by the browser, and will always be blocked on mobile browsers.
- It therefore makes sense to design your web page / application to not rely on play-on-load. You should always provide another way for the user to initiate the verbal interaction with your speaking character, in case play-on-load is blocked.

Special note regarding page activation on the Safari browser:

> As mentioned above, when a user interacts with your page, by clicking (or touching) anywhere on the page, the browser considers the page "activated" – and media playback is henceforth enabled. That seems to be true for every browser except Safari.
>
> Safari is stricter, and looks for direct intent by the user to initiate media play. In other words, if the user clicks on a button that directly initiates speech, Safari is satisfied. But if user interaction does not initiate speech, and sometime later an API call attempts to initiate speech unprompted by user interaction, that speech would be blocked by Safari, as it would not consider the page to be "activated".
>
> This different approach by Safari does not usually require any special attention, but in some cases it might. An example we sometimes come across is using Speech to Text in the browser for dialog with your SitePal character. Initial user interaction with the page which activates the microphone (for example) does not satisfy Safari.
>
> A great example on our support page demonstrates how this problem can be resolved.
> See - https://sitepal.com/api/examples/sayAI-STT.html
>
> Check out the source code and note how a click on "Start Listening" makes a call to our saySilent(0) API call – which generates (silent) audio playback and thus primes the Safari browser to accept future speech API calls.
> We are not aware of any browser other than Safari that requires this treatment.

Special note regarding page activation on Javascript Frameworks – React, Angular & Vue

> In JS Frameworks the saySilent page activation should be done for all browsers and platforms (not only mobile). To activate the page you need to call -
>
> ```
> window.saySilent(0)
> ```
>
> and NOT -
>
> ```
> saySilent(0)
> ```
>
> Calling the latter is not always successful - adding the "window" prefix always is. Make the call the first time the user clicks on any button. There is no need to call it again in the same page session. See our React, Angular and Vue examples (available on our support page) – review their source code as reference.

Programming for Mobile

This API is fully compatible with all major browsers on both desktop and mobile (the term 'desktop' is used here to refer to non-mobile client side environments, such as desktop and laptop computers of all types). This means that you need not do anything different in order to support mobile browsers when using this API.

> *Note: the function 'setPlayerVolume' does not have any effect in some mobile browsers – but there is no harm in making the call.*

Responsive Design

When embedding your Scene in a page which is designed to be responsive, turn ON the ‘Responsive’ attribute when publishing your Scene. This will cause the Scene to automatically resize itself to the container in which it is embedded. In such a case, the Scene’s specified dimensions are ignored, and no programming is required.

If, however, you prefer to finetune responsive behavior yourself, you have the option of using the "dynamicResize" API function to adjust the Scene’s dimensions. In such a case the ‘Responsive’ attribute should be kept OFF when publishing.

Examples for both use cases are available on our support examples page. See reference links in Appendix A.

Embedding in React, Angular, Vue & NextJS

JavaScript framesowrks have become quite popular and can be a great way to develop your web site or web application. Because of the unique ways in which web pages are loaded in these frameworks, we’ve made some adjustments to make our embed code compatible with them.

To embed your SitePal character in React, Angular, Vue & NextJS you will need to follow the instructions provided on our support page. See “Embed in JavaScript Frameworks” section here: www.sitepal.com/support.

Belated Loading of Your Scene

In some cases it may be useful or even necessary to embed your Scene in a web page, without loading it when the page loads. For example you may want to display your avatar in a pop-up or other UI element that is not immediately displayed, but is technically part of the same page.

To accomplish this, include your embed code on your page, in the appropriate place, but set the embed code ‘load’ parameter to 0 (see Embed Code specification below). Your Scene will not load and nothing will be displayed, but the embed code will be lodged in your page waiting for instructions. To load your Scene, use the ‘loadSceneByID’ API function, and its counterpart ‘unloadScene’ to achieve the opposite effect. API examples on our support page demonstrate this functionality.

Portals & Embedding Multiple Characters on a Page

Yes, you may embed multiple characters in a page. But how do you address an API call to a specific character when there are several on the page? To do so we introduce the concept of ‘Portal’. A Portal is what we call the embed code placed in your page, separate from the Scene embedded in it. As you will see, using portals makes it possible to load a different Scene to replace a Scene in your page, and also enables API targeting.

Calling the function -

```
selectPortal(a);
```

directs all subsequent API calls to the indicated Portal. If your page contains only one Scene, you can safely ignore all this.

Please see the documentation for ‘selectPortal’ for details. We also put together API examples demonstrating how to use ‘selectPortal’ to implement a conversation between two characters on your page. Check them out on our support page, in the ‘Advanced API Examples’ section.

About the Embed Code

The embed code function, which is generated automatically as part of your Scene's embed code, is not part of this API per se, as it is not designed to be called directly. Don't. Furthermore, the syntax of the embed code and the meaning & values of parameters may change in future (though it will always be backwards compatible).

For these reasons, it is not necessary to understand the details of the embed code syntax in most cases, but we provide the following as a matter of record.

Note: the parameters 'load' and 'context' are the only parameters you may need to set manually.

```
AC_Vhost_Embed(
        accountID,          // your account ID
        height,             // Scene height
        width,              // Scene width
        bgcolor,            // RGB hex value of embed background color
        firstscene,         // Scene to be loaded – reserved, always use 1
        controls,           // display controls: 0 – never; 1 – as needed; 2 - always
        ss,                 // Scene ID,
        sl,                 // Slide ID – reserved, always use 0.
        load,               // Load the Scene: 0 – do not load; 1 – load;
                            // Delay the loading of the Scene. See 'Belated Loading' section
        context,            // 0 – Standard web page; 1 – JS Framework
                            //       Set to '1'  if loading Scene into JS Frameworks
        embedId,            // unique string identifies this embed instance
        version,            // always use 0
        responsive,         //  0 – Standard behavior; 1 – Responsive behavior
        os)                 // overlay string – provided only with overlay embed
```

Error Handling

This API is designed to fail silently, with error messages written to the console, with one exception:

a Licensed Domain* infraction is the only case where a JS alert message will popup in your page in case of failure.

** Note: All speech generating functions require that your web page domain be pre-specified – referred to as a "Licensed Domain". This is a security precaution. Licensed domains can be added/edited in your account's "Settings" page.*

In addition to console logging, all speech generating functions also return status as a Json object with the following values:

`id`	Audio request unique ID.- 12 digit alphanumeric string
`status`	0 – normal completion
	1 – an error has occurred
	2 – blocked* by browser; user did not interact with page.

**Note: see "Play on Load" section in the Introduction for an explanation of 'blocked' status.*

`message` Error message – in case of error

Example: `{id: 'xxxxxxxxxxxx', status: 0, message: '' }`

This status is returned by the following speech generating functions:

- `sayAudio`
- `sayText`
- `loadAudio`
- `loadText`
- `sayAI`

The 'sayAI' function, also returns the status in the vh_aiResponse callback.

The vh_audioError callback:

The 'vh_audioError' callback allows you to improve error handling & logging for speech generating functions. Using it is optional.

This callback is needed because sayText (and any of the other speech functions listed above) is asynchronous unless you explicitly use 'await'.
So when 0 is returned, it does not mean that the audio was successfully processed. All it means is that your request was received, parameters appear to be valid, it was assigned an audio ID & submitted for processing. At that point, we don't yet know whether the audio will process without error.

After processing - If the audio is good, we call `vh_audioStarted`; if not (or a timeout), we call `vh_audioError`. The Audio iD is delivered by the callback in each case.

So unless you use 'await' (and forego asynchronous execution) – there is no way for you to know whether audio processing was successful from the function's return value alone. You need to listen to to both 'vh_audioStarted', and 'vh_audioError' – one is called in case of success, the other in case of failure.

Function Reference

Animation Control Functions

followCursor(mode)

Turn “follow cursor” to the OFF, ON IN BOX, or ON IN PAGE state. If OFF, the character’s gaze 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 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 ()

Toggle between the frozen and normal states. When frozen – all character movement stops. If the character is speaking, speech is paused. Unfrozen - character wakes up. If the character was previously paused in mid-speech, speech resumes from that point.

Arguments:

None.

Example:

```
freezeToggle()
```

---

recenter()

Cause the character to set its gaze to the default, centered position.

Arguments:

None.

Example:

```
recenter()
```

---

setGaze(degrees, duration, [amplitude])

Set the direction & amplitude of the character's head and eye movement.

This call will cause the 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 character is requested to speak.

The optional *amplitude* 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.
`amplitude`	Optional. Numeric. In percent. 0-100. Default = 100.

Example:

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

---

setFacialExpression(expression, amplitude, duration)

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. setFacialExpression 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

"None"	neutral - the default expression. Other parameters are ignored.
"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.

amplitude — Numeric. Range: 0, 1.0

The extent to which the expression should be applied. Using higher values then 1.0 is not blocked, and might be useful in some cases - but can lead to unexpected results. Feel free to experiment.

duration — Required if expression!="None", Integer.

Time in seconds. Use -1 for indefinite duration.

Example:

```
setFacialExpression("OpenSmile", 0.8, 5);
                                    // sets expression to OpenSmile for 5 seconds
                                    // at 80% amplitude
```

---

clearExpressionList( )

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

Clear all expressions.

Example:

```
clearExpressionList();
```

---

setIdleMovement(frequency, [amplitude])

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

Characters that are not engaged in speaking, following the cursor, or gazing (via the *setGaze* api) 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` — Required. 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)

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);
```

---

setBlinking(frequency)

Note: This function is only supported for 3D characters.
This function enables users to set the frequency of the character's eye blinking, or disable blinking altogether.

Arguments:

`frequency`	Required. Numeric. The blinking frequency. Lower value means less frequent blinking. Values are 0 to 100, corresponding to an average blink interval of 0.5 sec to 10 sec. The default value is 75, corresponding to an average blink interval of 3 sec. Use 0 to turn off blinking.

Example:

```
setBlinking(50);
```

---

getFPS(mode)

Note: This function is only available to the Platinum and Integrator Plans.

Get the GPU refresh rate for either active or frozen mode. The function returns a float value. GPU render frequency is different when the avatar is active vs. frozen. The default values used are 24 fps and 2 fps respectively.

Arguments:

`mode`	Optional. Numeric. 0 – active (default). 1 – frozen.

Return value:

Float. The requested refresh rate in frames per second (fps) is returned.

Examples:

```
fps_active = getFPS(0);
fps_freeze = getFPS(1);
```

setFPS(rate, mode)

Note: This function is only available to the Platinum and Integrator Plans.

Set the GPU refresh rate for either active or frozen mode. GPU render frequency is different when the avatar is active vs. frozen. The default values used are 24 fps and 2 fps respectively.

Arguments:

`rate`	Required. Float. The specified refresh rate in frames per second.
`mode`	Optional. Numeric. 0 – active (default). 1 – frozen.

Examples:

```
setFPS(15);        // set the active rate to 15 fps
setFPS(1.5,1);     // set the frozen rate to 1.5 fps
```

Speech Functions

loadAudio(name)

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.

Return Value:

Json object with 3 fields: 'id', 'status' and 'message'. See Error Handling for details.

Example:

```
err = loadAudio('audioname')
```

---

loadText(txt,voice,lang,engine,[effect], [effLevel],[xData1],[xData2])

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 load. Text is limited to 10,000 characters
`voice`	Required. String – 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" – Durationlevels: -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.

`xData1` Optional. String.

xData parameters are used to enable engine specific attributes when applicable. This is currently used by Engines 14 (Eleven Labs) & 15 (OpenAI). Values added for any other engine are ignored.

For engine 14 – if no value is provided, the following are used as defaults:

```
model_id=eleven_flash_v2_5,optimize_streaming_latency=2,
stability=0.5,similarity_boost=0.5,style=0,use_speaker_b
oost=true
```

For engine 15 – if no value is provided, the following are used as defaults:

```
model=tts-1,speed=1.0
```

If any of the above name-value pairs are NOT provided, default values are used. Review 3rd Party documentation for details about these attributes, and feel free to experiment.

`XData2` Optional. String. Reserved.

Return Value:

Json object with 3 fields: 'id', 'status' and 'message'. See Error Handling for details.

Example:

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

---

sayAudio(name, [startTime])

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.

Return Value:

Json object with 3 fields: 'id', 'status' and 'message'. See Error Handling for details.

Example:

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

---

sayText (txt,voice,lang,engine,[effect], [effLevel],[xData1],[xData2])

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

For detailed step by step instructions, please review Using the TTS API in the support section.

Note: This function is available only to the Silver plan or higher and will work only within a specified licensed domain. Domain specific licensing is a security measure. If the account is not TTS enabled, or the Scene is embedded within a non-licensed domain, then this call will generate an alert. To edit your licensed domains login to your account and goto the "Settings" page, available from the 'Welcome' menu at top right.

Arguments:

`txt` Required. String - The text to speak. Text is limited to 10,000 characters.
Note that strings longer than 900 characters will consume multiple streams when played, in integer multiples of 900. Example: a string 1500 characters long will consume 2 streams.

`voice` Required. String – 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.

`xData1` Optional. String.

xData parameters are used to enable engine specific attributes when applicable. This is currently used by Engines 14 (Eleven Labs) & 15 (OpenAI). Values added for any other engine are ignored.

For engine 14 – if no value is provided, the following are used as defaults:

```
model_id=eleven_flash_v2_5,optimize_streaming_latency=2,
stability=0.5,similarity_boost=0.5,style=0,use_speaker_b
oost=true
```

For engine 15 – if no value is provided, the following are used as defaults:

```
model=tts-1,speed=1.0
```

If any of the above name-value pairs are NOT provided, default values are used. Review 3rd Party documentation for details about these attributes, and feel free to experiment.

`XData2` Optional. String. Reserved.

Return Value:

Json object with 3 fields: 'id', 'status' and 'message'. See Error Handling for details.

Examples:

```
err = sayText('Hello World',3,1,3)
err = sayText('Hello World',3,1,3,'S',-2)
```

---

sayAI(txtQ,voice,lang,engine,[effect], [effLevel],[botVendor],[botName], [resLength],[xData1],[xData2])

```
Note: This function is available only to Gold plan and higher accounts
and will only work within a pre-specified licensed domain. If this
function call is used from a non-licensed domain, the call will generate
an alert. You can edit your licensed domains in your account's "Settings"
page.
```

For detailed step by step instructions on getting started with AI using this function, please review Implementing Your AI Agent in the Support section.

An Artificial Intelligence knowledge base provides a real time audio & text 'response' to a text 'question'. The response Audio is generated & spoken according to the selected voice. The response text is also returned via the event callback function 'vh_aiResponse()', in case you want to display it.

```
Note: If you have no need for the text response, there is no need to
setup the callback function.
```

The response is returned from your specified bot. If you do not specify botVendor and botID, the default would be the first AIMC bot in your account. Every SitePal Gold plan and higher comes with a default bot that you may edit in AIMC. Even if you have not yet edited your bot, it is available to your account.

```
Note: The default knowledge base is based on the extensive A.L.I.C.E.
AIML set, which includes over 23,000 data entries (available in English
only). This knowledge base can be edited and customized in the AI
Management Center (AIMC) – click on AIMC from the main menu in your
account.
```

SitePal is also pre-integrated with select third party AI vendors, allowing you to connect your SitePal speaking character with your 3rd party bot. To do so, specify BotVendor, and BotName. See parameter documentation below.

```
To use a 3rd party bot, you need to setup the botName & 3rd party API Key
in your account's "Connect" page.
```

Arguments:

`txtQ`	Required. String - The text question or input.
`voice`	Required. String – 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.

`BotVendor` Optional.

- Built-in AIMC, leave blank.
- Built-in SitePal AI, use: 'SP' (placeholder - not yet launched)
- Chatbase, use: 'CB'
- ChatGPT, use: 'GPT'
- Google Dialog Flow, use: 'DF'
- Pandorabots, use: 'PB'

`BotName` For built-in AIMC, leave blank.

For others you must use the bot name as specified in your "Connect" page.

`resLength` Optional. Integer. Max length of response, in words. Default value:160.

`xData1` Optional. String.

xData parameters are used to enable engine specific attributes when applicable. This is currently used by Engines 14 (Eleven Labs) & 15 (OpenAI). Values added for any other engine are ignored.

For engine 14 – if no value is provided, the following are used as defaults:

```
model_id=eleven_flash_v2_5,optimize_streaming_latency=2,
stability=0.5,similarity_boost=0.5,style=0,use_speaker_b
oost=true
```

For engine 15 – if no value is provided, the following are used as defaults:

```
model=tts-1,speed=1.0
```

If any of the above name-value pairs are NOT provided, default values are used. Review 3rd Party documentation for details about these attributes, and feel free to experiment.

`XData2` Optional. String. Reserved.

Return Value:

Json object with 3 fields: 'id', 'status' and 'message'. See Error Handling for details.

Examples:

```
err = sayAI('Sing me a song',3,1,3)
err = sayAI('Sing me a song',3,1,3,'P',-1)
err = sayAI('Sing me a song',3,1,3,'','','PB','rosie')
```

---

sayAIResponse

This function has been deprecated. Please use *sayAI*.

saySilent (seconds)

Speech is visually simulated and a silent audio is played..

This function call may be useful when you want to call attention to your character but cannot use audio to do so. A pertinent example might be use in ad banners, where the use of audio may only be enabled after user interaction.

Another reason to use this function would be to “activate” a page to accept media playback API calls. To do so, call this function with parameter value 0. See special note in “Play on Load” section in the introduction for more information & and example of this use case.

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)      // silently animate the mouth for 10 seconds
saySilent(0)       // no mouth animation takes place
```

---

setPlayerVolume (level)

Note: This function has no effect on some mobile browsers.
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 ()

Stop the speech of a currently speaking 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 ()

Plays or replays current Scene, from the start.

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

Arguments:

None.

Example:

```
replay()
```

Scene Attributes

getSceneAttributes()

Retrieve several key Scene attributes. Function returns an object.

Arguments:

`None.`

Return Values:

An object with following named values is returned:

`sceneID`	Scene's database ID
`bgName`	Logical name of background assigned to Scene if any.
`audioName`	Logical name of audio assigned to Scene if any.
`modelID`	Database ID of model assigned to Scene if any.

Example:

```
var attr = getSceneAttributes();
var sceneID       = attr.sceneID;
var bgName        = attr.bgName;
var audioName     = attr.audioName;
var modelID       = attr.modelID;
```

---

setBackground (bgName)

Scene Background is modified. Change is not persistent.

Arguments:

`bgName`	Required. String. The logical name of the background as specified within the account. If an empty string is provided, the background is cleared.

Example:

```
setBackground('background name');
setBackground('');                    // clear the bg
```

---

setBackgroundColor (bgColor)

Scene background color is modified. Change is not persistent.

Note: If the Scene has a background image, the background color, which is displayed behind the image, may not be visible, except while loading.

Arguments:

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

Example:

```
setBackgroundColor('0000AA');
```

---

setColor (part,color)

Dynamically modify the color of the specified character "part".

Colors are applied to the grayscale baseline of the specific character's design. Therefore, the effect of a color on the specified area may not exactly match its color value, as you are seeing the effect of its application to a non-white surface. For the same reason, results may differ when the same color is applied to different characters.

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')
```

---

setStatus (interruptMode,progressInterval,gazeSpeed,displayControls,enableLog)

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

`displayControls`

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

Controls the display of playback controls.

0 – Never. This can be useful if you want to create your own controls
1 – As needed. Controls are shown when cursor rolls over the Scene (or Scene is touched)
2 – Always

`enableLog`

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

0 – only minimal logging is displayed in console

1 – more detailed logging is displayed

Example:

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

---

dynamicResize (width, height)

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

Note that the simplest way to embed your SitePal avatar in a responsve page is to use the "responsive" attribute when publishing your character (this option is available when you copy the embed code). Use this function to implement your own custom resizing only if you have special requirements unmet by our default behaviour.

When calling this function there is no requirement to maintain the original aspect ratio. If you would like to retain the relative position of the character within the Scene frame, you should retain the aspect ratio. Otherwise, character will be re-positioned as best possible.

Tip: To control the dimensions of the embedded Scene when page is initially loaded, you could set the width and height *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, sceneid,
0,1,0,'94fb29b9a4767343f36dd16fc8c0f81a',0);</script>
```

Arguments:

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

Example:

```
dynamicResize(300,200)
```

---

getAudioObject ()

Note: This function is only available to the Platinum and Integrator Plans.

This function returns a handle to the current Javascript audio object, if one exists.

It is best used in the context of the callback vh_audioStarted. If this function is called when no audio is playing - 'null' is returned.

Note that -

- the function returns the actual handle to the playing audio.
- the object should be handled as 'read only' - any changes can introduce problems.
- the object is only valid during playback and may become invalid immediately after.

Arguments:

None.

Example:

```
var audioObject = getAudioObject();
```

---

is3D ()

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 that are embedded as an overlay on top of the page. If any of these functions is called for a Scene which is embedded in-line within page content, the function call will have no effect.

overlayOpen (mode, play)

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

Arguments:

`mode` `'max' - open Scene in maximized mode or toggle to maximize mode.` If the Scene is already maximized, this call has no effect.

`'min' - open Scene 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 ()

Scene 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()
```

Manage Embeded Scenes in Page

Note: The word “Scene”, as used in this section, should be understood to refer to the ‘Embedded Entity’ be it a Scene.

loadSceneByID (sceneID, slideID)

Replace the embedded Scene with another. The current Scene is interrupted, and the specified Scene is loaded instead.

Arguments:

`sceneID`	*Optional.* ID the Scene to load. If not provided, the Scene which is specified in the embed code is loaded. In your embed code, the Scene ID is parameter #7. To find the Scene ID for the Scene you want to load, select the Publish in Web Page option for that Scene, and locate the value used as the 7th parameter. See “About the Embed Code” for more information. Note: sceneID must belong to your account.
`slideID`	*Optional.* Reserved, do not use.

Example

```
loadSceneByID()          – load Scene specified in the portal's embed code
loadSceneByID(459284)    – load specified Scene.
```

---

unloadScene()

Unload the embedded Scene. This removes the Scene from the page, and retains only the embed portal. Another (or the same) Scene may be subsequently loaded into the portal.

Arguments:

none.

Example

```
unloadScene()
```

---

selectPortal (portal)

Notes:

- The “SitePal Conversation” example in the Advanced Examples page in the Support section nicely demonstrates how this function is used.
- Using this function requires embed functions v.4. If you are using a lower version of the embed code, republish and replace the embed code in your page with a fresh version.

A “portal” is what we call the space on the page created by the embed code, to be used for loading a Scene into. Typically the portal loads and displays a Scene which was originally specified when the embed code was created. But it is possible to load another Scene into it, to replace a previously loaded Scene. Hence the name: portal.

When multiple portals are embedded on the same page, only one can accept API calls at any given moment. This call is used to select the embed portal that accepts API calls, and thus allow the API to target a specific portal.

To use this call, you must collect the embed portal reference when the page is loaded – by assigning the values returned by the “Embed” function as follows:

```
var alice = AC_Vhost_Embed(…
var bob = AC_Vhost_Embed(…
```

The variables ‘alice’ and ‘bob’ in this example will contain the reference values for the two portals. Use these values to specify the currently active portal, and thus target your API functions. There is no limit to the number of portals you can embed on a single page.

Arguments:

`portal`	Required. Portal reference retrieved from Embed function.

Example:

```
selectPortal(alice)
```

Status Callback Functions

Callback Functions can help improve coordination between the embedded Scene and your page / application.

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 using callback functions is optional; There is no need to add callback functions which you do not intend to use.

vh_sceneLoaded (index, portal, errCode, errMsg)

Triggered when the Scene is fully loaded & displayed. Use this callback to verify Scene is ready to accept API calls.

Note: *SitePal API functions will only work after your Scene has completed loading. It is therefore advisable to always implement the "vh_sceneLoaded" callback & check that it has been called before attempting to call any API function.*

Arguments:

`index`	Deprecated, ignore
`portal`	Reference to the portal that generated the call
`errCode`	0 – normal completion
	1 – failed to load: account inactive or Scene missing
	2 – Scene loaded into (0,0) container – not visible
`errMsg`	Error message – provided in case of error

Example -

```
function vh_sceneLoaded(index,portal,code,msg){
      alert("scene has loaded");
}
```

---

vh_talkStarted (portal)

Triggered when the character starts talking. When several audios are played in sequence, this callback will be dispatched at the start of the sequence.

Arguments:

`portal`	Reference to the portal that generated the call.

Example -

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

vh_talkEnded (portal)

Triggered when the character is done talking. When several audios are played in sequence, this callback will be dispatched at the end of the sequence.

Arguments:

`portal`	Reference to the portal that generated the call.

Example -

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

---

vh_audioStarted (audID, portal)

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

Arguments:

`portal`	Reference to the portal that generated the call.

Example -

```
function vh_audioStarted(audID, portal){
}
```

---

vh_audioEnded (audID, portal)

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

Arguments:

`portal`	Reference to the portal that generated the call.

Example -

```
function vh_audioEnded(audID, portal){
}
```

---

vh_audioError (audID, portal, errCode, errMsg)

Triggered when an error prevents audio playback from starting. When this callback is called, vh_audioStarted will not be called.

Arguments:

`audID`	The ID of the audio that failed.
`portal`	Reference to the portal that generated the cal to play the audio.
`ErrCode`	TBD
`errMsg`	TBD

Example -

```
function vh_audioError(id,portal,code,msg){
      alert("audio " + id + " has failed");
}
```

---

vh_playPause (status, portal)

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

Arguments:

`status`	0=paused; 1=playing.
`portal`	Reference to the portal that generated the call.

Example - JavaScript

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

---

vh_audioProgress (percentPlayed, portal)

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.
`portal`	Reference to the portal that generated the call.

Example -

```
function vh_audioProgress(percentPlayed, portal) {
}
```

---

vh_aiResponse (responseText, portal, status)

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:

`responseText`	Response text. If Status indicates an error, the error description is returned.
`portal`	Reference to the portal that generated the call.
`status`	0 – normal completion 1 – an error has occurred 2 – blocked* by browser; user did not interact with page. **Note: see "Play on Load" section in the Introduction for an explanation of 'blocked' status.*

Example -

```
function vh_aiResponse(responseText, portal, status){
}
```

---

vh_audioLoaded (audioName, portal)

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

Arguments:

`audioName`	Loaded audio name
`portal`	Reference to the portal that generated the call.

Example -

```
function vh_audioLoaded(audioName,portal){
}
```

---

vh_ttsLoaded (audioText, portal)

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

Arguments:

`audioText`	Loaded text to be spoken
`portal`	Reference to the portal that generated the call.

Example -

```
function vh_ttsLoaded(audioText, portal){
}
```

---

vh_portalReady (portal)

Triggered when the embed code is fully loaded, and before Scene is to be loaded. Use this callback to verify that the portal is ready to accept loadScene calls. No other API calls can be made at this time.

Note: when this callback is received, your Scene has not yet loaded. You must wait for the vh_sceneLoaded callback before calling any API function which affects the Scene.

Arguments:

`portal`	Reference to the portal that generated the call.

Example -

```
function vh_portalReady(portal){
      alert("the embed code has loaded");
}
```

Appendix A: API Examples

We've put together a comprehensive collection of technical examples that demonstrate how to use every one of our API functions and callbacks. A number of advanced use cases are also demonstrated.

As you check out these examples, please feel free to review their source code, and to copy that source code for use in your own pages as you get started. You will of course need to replace the embed code in the example with embed code from your own account, to have control over Avatar design & speech.

Note: a common problem when getting started is not setting up a "Licensed Domain" - which is required for certain functions to work. To add or edit your Licensed Domain(s) goto your "Settings" page.

Our API support examples can be found here:

Using the Client API - Technical Examples

Additional reference material can be found in our support section.

Appendix B: TTS Languages and Voices

The Text-To-Speech languages and voices available with SitePal.are listed here:

SitePal TTS Available Languages & Voices

This reference document lists the TTS languages and voices separated into two main sections:

- Built-in Voices
- 3rd Party Voices

The built-in TTS voices are available to SitePal customers with every subscription plan that supports TTS.

3rd party voices are available from leading providers which have been pre-integrated with SitePal. To use 3rd party voices you need to *connect* them to your SitePal account. You can do so in your SitePal 'Connect' page.

Detailed instructions for setting up connectivity with each of the available 3rd party providers are available as well.

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 #2 & #4 do not support SSML tags. Please select only voices from Engines #3 & #7 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*

Tip : A great environment in which to test your SSML Tags with different voices is available on our support page – look for "Fine Tuning Text-To-Speech". This test page lets you select a voice and provides handy samples of the most popular SSML tags to try out.

Only voices that support SSML tags are available to select – this can be a time saver

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".