pyttsx Documentation

Release 1.0

Peter Parente

December 30, 2012

CONTENTS

This documentation describes the pyttsx Python package v 1.0 and was rendered on December 30, 2012.

Table of Contents

CHAPTER

ONE

SUPPORTED SYNTHESIZERS

Version 1.0 of pyttsx includes drivers for the following text-to-speech synthesizers. Only operating systems on which a driver is tested and known to work are listed. The drivers may work on other systems.

- SAPI5 on Windows XP and Windows Vista
- NSSpeechSynthesizer on Mac OS X 10.5 (Leopard) and 10.6 (Snow Leopard)
- espeak on Ubuntu Desktop Edition 8.10 (Intrepid), 9.04 (Jaunty), and 9.10 (Karmic)

The `pyttsx.init()` documentation explains how to select a specific synthesizer by name as well as the default for each platform.

CHAPTER

TWO

USING PYTTSX

An application invokes the `pyttsx.init()` factory function to get a reference to a `pyttsx.Engine` instance. During construction, the engine initializes a `pyttsx.driver.DriverProxy` object responsible for loading a speech engine driver implementation from the `pyttsx.drivers` module. After construction, an application uses the engine object to register and unregister event callbacks; produce and stop speech; get and set speech engine properties; and start and stop event loops.

2.1 The Engine factory

`pyttsx.`**`init`**(*[driverName : string, debug : bool]*) → pyttsx.Engine

Gets a reference to an engine instance that will use the given driver. If the requested driver is already in use by another engine instance, that engine is returned. Otherwise, a new engine is created.

Parameters

- **driverName** – Name of the `pyttsx.drivers` module to load and use. Defaults to the best available driver for the platform, currently:
  - *sapi5* - SAPI5 on Windows
  - *nsss* - NSSpeechSynthesizer on Mac OS X
  - *espeak* - eSpeak on every other platform
- **debug** – Enable debug output or not.

Raises

- **ImportError** – When the requested driver is not found
- **RuntimeError** – When the driver fails to initialize

2.2 The Engine interface

class `pyttsx.engine.`**`Engine`**

Provides application access to text-to-speech synthesis.

`connect`(*topic : string, cb : callable*) → dict

Registers a callback for notifications on the given topic.

Parameters

- **topic** – Name of the event to subscribe to.

- **cb** – Function to invoke when the event fires.

Returns A token that the caller can use to unsubscribe the callback later.

The following are the valid topics and their callback signatures.

started-utterance
Fired when the engine begins speaking an utterance. The associated callback must have the folowing signature.

onStartUtterance(*name : string*) → None
Parameters **name** – Name associated with the utterance.

started-word
Fired when the engine begins speaking a word. The associated callback must have the folowing signature.

onStartWord(*name : string*, *location : integer*, *length : integer*)
Parameters **name** – Name associated with the utterance.

finished-utterance
Fired when the engine finishes speaking an utterance. The associated callback must have the folowing signature.

onFinishUtterance(*name : string*, *completed : bool*) → None
Parameters
- **name** – Name associated with the utterance.
- **completed** – True if the utterance was output in its entirety or not.

error
Fired when the engine encounters an error. The associated callback must have the folowing signature.

onError(*name : string*, *exception : Exception*) → None
Parameters
- **name** – Name associated with the utterance that caused the error.
- **exception** – Exception that was raised.

disconnect(*token : dict*)
Unregisters a notification callback.

Parameters **token** – Token returned by `connect()` associated with the callback to be disconnected.

endLoop() → None
Ends a running event loop. If `startLoop()` was called with *useDriverLoop* set to True, this method stops processing of engine commands and immediately exits the event loop. If it was called with False, this method stops processing of engine commands, but it is up to the caller to end the external event loop it started.

Raises RuntimeError When the loop is not running

getProperty(*name : string*) → object
Gets the current value of an engine property.

Parameters **name** – Name of the property to query.

Returns Value of the property at the time of this invocation.

The following property names are valid for all drivers.

rate
Integer speech rate in words per minute. Defaults to 200 word per minute.

voice
String identifier of the active voice.

voices
List of `pyttsx.voice.Voice` descriptor objects.

volume
Floating point volume in the range of 0.0 to 1.0 inclusive. Defaults to 1.0.

isBusy() → bool
Gets if the engine is currently busy speaking an utterance or not.

Returns True if speaking, false if not.

runAndWait() → None
Blocks while processing all currently queued commands. Invokes callbacks for engine notifications appropriately. Returns when all commands queued before this call are emptied from the queue.

say(*text : unicode*, *name : string*) → None
Queues a command to speak an utterance. The speech is output according to the properties set before this command in the queue.

Parameters

- **text** – Text to speak.
- **name** – Name to associate with the utterance. Included in notifications about this utterance.

setProperty(*name*, *value*) → None
Queues a command to set an engine property. The new property value affects all utterances queued after this command.

Parameters

- **name** – Name of the property to change.
- **value** – Value to set.

The following property names are valid for all drivers.

rate
Integer speech rate in words per minute.

voice
String identifier of the active voice.

volume
Floating point volume in the range of 0.0 to 1.0 inclusive.

startLoop([*useDriverLoop : bool*]) → None
Starts running an event loop during which queued commands are processed and notifications are fired.

Parameters **useDriverLoop** – True to use the loop provided by the selected driver. False to indicate the caller will enter its own loop after invoking this method. The caller's loop must pump events for the driver in use so that pyttsx notifications are delivered properly (e.g., SAPI5 requires a COM message pump). Defaults to True.

stop() → None
Stops the current utterance and clears the command queue.

2.3 The Voice metadata

class `pyttsx.voice.`**Voice**
Contains information about a speech synthesizer voice.

age
Integer age of the voice in years. Defaults to `None` if unknown.

gender
String gender of the voice: *male*, *female*, or *neutral*. Defaults to `None` if unknown.

id
String identifier of the voice. Used to set the active voice via `pyttsx.engine.Engine.setPropertyValue()`. This attribute is always defined.

languages
List of string languages supported by this voice. Defaults to an empty list of unknown.

name
Human readable name of the voice. Defaults to `None` if unknown.

2.4 Examples

2.4.1 Speaking text

```
import pyttsx
engine = pyttsx.init()
engine.say('Sally sells seashells by the seashore.')
engine.say('The quick brown fox jumped over the lazy dog.')
engine.runAndWait()
```

2.4.2 Listening for events

```
import pyttsx
def onStart(name):
   print 'starting', name
def onWord(name, location, length):
   print 'word', name, location, length
def onEnd(name, completed):
   print 'finishing', name, completed
engine = pyttsx.init()
engine.say('The quick brown fox jumped over the lazy dog.')
engine.runAndWait()
```

2.4.3 Interrupting an utterance

```
import pyttsx
def onWord(name, location, length):
   print 'word', name, location, length
   if location > 10:
```

```
      engine.stop()
engine = pyttsx.init()
engine.say('The quick brown fox jumped over the lazy dog.')
engine.runAndWait()
```

2.4.4 Changing voices

```
engine = pyttsx.init()
voices = engine.getProperty('voices')
for voice in voices:
   engine.setProperty('voice', voice.id)
   engine.say('The quick brown fox jumped over the lazy dog.')
engine.runAndWait()
```

2.4.5 Changing speech rate

```
engine = pyttsx.init()
rate = engine.getProperty('rate')
engine.setProperty('rate', rate+50)
engine.say('The quick brown fox jumped over the lazy dog.')
engine.runAndWait()
```

2.4.6 Changing volume

```
engine = pyttsx.init()
volume = engine.getProperty('volume')
engine.setProperty('volume', volume-0.25)
engine.say('The quick brown fox jumped over the lazy dog.')
engine.runAndWait()
```

2.4.7 Running a driver event loop

```
engine = pyttsx.init()
def onStart(name):
   print 'starting', name
def onWord(name, location, length):
   print 'word', name, location, length
def onEnd(name, completed):
   print 'finishing', name, completed
   if name == 'fox':
      engine.say('What a lazy dog!', 'dog')
   elif name == 'dog':
      engine.endLoop()
engine = pyttsx.init()
engine.say('The quick brown fox jumped over the lazy dog.', 'fox')
engine.startLoop()
```

2.4.8 Using an external event loop

```
engine = pyttsx.init()
engine.say('The quick brown fox jumped over the lazy dog.', 'fox')
engine.startLoop(False)
# engine.iterate() must be called inside externalLoop()
externalLoop()
engine.endLoop()
```

CHAPTER

THREE

IMPLEMENTING DRIVERS

You can implement new drivers for the `pyttsx.Engine` by:

1. Creating a Python module with the name of your new driver.
2. Implementing the required driver factory function and class in your module.
3. Using methods on a `pyttsx.driver.DriverProxy` instance provided by the `pyttsx.Engine` to control the event queue and notify applications about events.

3.1 The Driver interface

All drivers must implement the following factory function and driver interface.

`pyttsx.drivers.`**`buildDriver`**(*proxy : pyttsx.driver.DriverProxy*) → pyttsx.drivers.DriverDelegate
Instantiates delegate subclass declared in this module.

Parameters **proxy** – Proxy instance provided by a `pyttsx.Engine` instance.

class `pyttsx.drivers.`**`DriverDelegate`**

Note: The `DriverDelegate` class is not actually declared in `pyttsx.drivers` and cannot server as a base class. It is only here for the purpose of documenting the interface all drivers must implement.

`__init__`(*proxy : pyttsx.drivers.DriverProxy*, **args*, ***kwargs*) → None
Constructor. Must store the proxy reference.

Parameters **proxy** – Proxy instance provided by the `buildDriver()` function.

`destroy`()
Optional. Invoked by the `pyttsx.driver.DriverProxy` when it is being destroyed so this delegate can clean up any synthesizer resources. If not implemented, the proxy proceeds safely.

`endLoop`() → None
Immediately ends a running driver event loop.

`getProperty`(*name : string*) → object
Immediately gets the named property value. At least those properties listed in the `pyttsx.Engine.getProperty()` documentation must be supported.

Parameters **name** – Name of the property to query.

Returns Value of the property at the time of this invocation.

say(*text : unicode*, *name : string*) → None
Immediately speaks an utterance. The speech must be output according to the current property values applied at the time of this invocation. Before this method returns, it must invoke `pyttsx.driver.DriverProxy.setBusy()` with value `True` to stall further processing of the command queue until the output completes or is interrupted.

This method must trigger one and only one *started-utterance* notification when output begins, one *started-word* notification at the start of each word in the utterance, and a *finished-utterance* notification when output completes.

Parameters

- **text** – Text to speak.
- **name** – Name to associate with the utterance. Included in notifications about this utterance.

setProperty(*name : string*, *value : object*) → None
Immediately sets the named property value. At least those properties listed in the `pyttsx.Engine.setProperty()` documentation must be supported. After setting the property, the driver must invoke `pyttsx.driver.DriverProxy.setBusy()` with value `False` to pump the command queue.

Parameters

- **name** – Name of the property to change.
- **value** – Value to set.

startLoop()
Immediately starts an event loop. The loop is responsible for sending notifications about utterances and pumping the command queue by using methods on the `pyttsx.driver.DriverProxy` object given to the factory function that created this object.

stop()
Immediately stops the current utterance output. This method must trigger a *finished-utterance* notification if called during on-going output. It must trigger no notification if there is no ongoing output.

After stopping the output and sending any required notification, the driver must invoke `pyttsx.driver.DriverProxy.setBusy()` with value `False` to pump the command queue.

3.2 The DriverProxy interface

The `pyttsx.drivers.buildDriver()` factory receives an instance of a `DriverProxy` class and provides it to the `pyttsx.drivers.DriverDelegate` it constructs. The driver delegate can invoke the following public methods on the proxy instance. All other public methods found in the code are reserved for use by an `pyttsx.Engine` instance.

class `pyttsx.driver.`**DriverProxy**

isBusy() → bool
Gets if the proxy is busy and cannot process the next command in the queue or not.

Returns True means busy, False means idle.

notify(*topic : string*, ***kwargs*) → None
Fires a notification.

Parameters **topic** – The name of the notification.

Kwargs Name/value pairs associated with the topic.

setBusy(*busy : bool*) → None
Sets the proxy to busy so it cannot continue to pump the command queue or idle so it can process the next command.

Parameters **busy** – True to set busy, false to set idle

CHAPTER

FOUR

CHANGELOG

4.1 Version 1.0

First release.

Project Links

- Project home page at Launchpad
- Package listing in PyPI
- Documentation in PyPI

PYTHON MODULE INDEX

p