Say It Smart Plugins

Similar to the ability for a developer to create custom elements, a developer can create their own Say It Smart plugins. A developer can produce plugins that handle brand new Say It Smart types as well as plugins that extend the functionality of existing Say it Smart plugins. Due to the integration requirements for Say It Smart plugins, they can be built only by using the Java API.

Custom Say It Smart plugins are integrated into both VXML Server and Call Studio as easily as Unified CVP Say It Smart plugins are. They can be deployed for a specific application or shared across all applications and are configured in the Studio in the same manner Unified CVP Say It Smart plugins are. With such seamless integration and effortless deployment, a developer can, over time, create entire libraries of custom Say It Smart plugins to use for their voice applications or potentially for resale.

This chapter describes in detail how to create custom Say It Smart plugins and integrate them into both Call Studio and VXML Server.

Design

Say It Smart plugins were designed to be very simple to build, extend, and deploy. Much of the design mirrors that of custom configurable elements, though Say It Smart plugins are simpler and set far fewer restrictions. Their sole purpose is to take input representing formatted data and convert it into a list of pre-recorded audio files with TTS backups and pauses if desired.

Similar to configurable elements, a Say It Smart plugin is constructed by creating a Java class that extends an abstract base class, `SayItSmartBase`. The base class defines abstract methods that must be implemented by the plugin to describe how Builder for Call Studio displays the plugin. A plugin that is to appear in the Builder must implement a Java marker interface named `SayItSmartPlugin`. Unlike elements, though, Say It Smart plugins have two methods to run, one for converting data to a set of audio files with TTS backups, and the other for converting the data using TTS only. These methods to run may throw a `SayItSmartException` that is used to indicate the inability of the plugin to convert the data passed to it.

The configuration of a Say It Smart plugin involves four options:

- The first is the Say It Smart type (such as phone number or date). Each type must be defined in a separate plugin class.

- The second option is the chosen input format. Input formats list how to expect the input data to arrive (such as a date with just the month and year or a date with the month, day and year). A plugin defines all the input formats it supports.
- The next option is the chosen output format. Output formats list how to render the converted data (such as reading back a time where 12:00 AM is read back as *noon* as opposed to *12 AM*). Output formats are dependent on input formats, so when the input format changes, the list of output formats available changes accordingly. The plugin defines these dependencies using a configuration method.
- The final option is the fileset. Filesets determine what group of audio files is used to render the same data (such as one that reads back a better-sounding number by requiring more audio files). Filesets are dependent on output formats, so when the output format changes, the list of filesets available changes accordingly. The plugin defines these dependencies using a configuration method.

Note A fileset deals with audio files so does not apply when the Say It Smart value is rendered in TTS only.

As usually occurs with components configured in Builder for Call Studio, each type, input format, output format, and fileset has a real name, display name, and description. The display name is shown in the Builder in dropdown menus. The real name is used everywhere else. This design allows the Say It Smart plugin developer to use a display name that visually represents the appropriate information but choose a real name that is small, easy to remember, and will not change. As long as the real name stays the same, the developer can change the display name of any component without affecting backwards compatibility. At this point, the descriptions are not displayed in the Builder and are there for future compatibility.

All Java classes related to Say It Smart plugins are found in the `com.audium.server.sayitsmart` package.

Note Unlike elements, Say It Smart plugin classes are instantiated as needed. This means that the developer is free to use static, member, and local variables as they would expect. You must avoid using static variables in Say It Smart plugin classes unless they are static final because static variables will be reset whenever the application is updated.

A checkbox titled **Remote Execution** is added to the **Say It Smart Plugin** for running it remotely. If the **Remote Execution** checkbox is enabled, it will use the URI values provided in the **Remote Url Settings** tab.

Methods to Run

The following points describe the methods to run:

-
  ```
  SayItSmartContent convertToFiles(Object data, String inputFormat,
                                   String outputFormat, String fileset)
  ```

  This is the method to run for converting data to a list of audio files with TTS backups. The first argument is the data to convert. This data can be any Java object and it is up to the plugin to cast to the appropriate type (usually depending on what the input format is). Most of the time, though, it will be considered a `String`. The next three arguments list the real names of the input format, output format, and fileset specified in the voice element configuration (either statically in Builder for Call Studio or dynamically).

The method returns an instance of `SayItSmartContent`. This class encapsulates any number of audio filenames, their TTS backups, and pauses to insert in the playback. The method to run must create an instance of this class, add the desired content to it, and return it. VXML Server then reads this content to generate the VoiceXML for the Say It Smart audio item in the audio group.

Note The path and file type options available in the Say It Smart configuration in the Builder are not passed here because they are added *automatically* by VXML Server once the plugin has converted the data. The plugin should produce just the audio file names without any paths or extensions.

Note While the option exists, the plugin need not include TTS backups for the audio files. They are used as a backup in case the audio file is not found or is corrupted. Not including a transcript (by making it `null`) would mean that if the audio file could not be found, the application would prematurely end with an error.

The developer can throw a `SayItSmartException` in this method indicating the data passed to the plugin could not be converted using the specified configuration. This would most likely end the call prematurely so the exception should be thrown only when the Say It Smart plugin cannot do what it is supposed to do and is unable to recover.

`convertToFiles()` is abstract, meaning every Say It Smart plugin must implement this method.

-
  ```
  SayItSmartContent convertToTTS(Object data, String inputFormat,
                                 String outputFormat, String fileset)
  ```

This is the method to run for converting data to a TTS string. The arguments are identical in this method as the c`onvertToFiles()` method.

Note Even though the fileset option technically does not apply here, it is included in case the plugin does alter the TTS output based on fileset information.

The method must still return a `SayItSmartContent` object since the plugin may still decide to play back the TTS content with pauses at certain points. In this case, the filename value would be set to `null`.

Note `convertToTTS()` is not abstract, `SayItSmartBase` actually provides a default implementation of this method. The default implementation simply calls the c`onverToFiles()` method, combines the TTS transcripts it receives and returns a new `SayItSmartContent` object with just those transcripts and any pauses. The plugin need only provide its own implementation of this method if the desired behavior is different.

The developer can throw a `SayItSmartException` in this method indicating the data passed to the plugin could not be converted using the specified configuration. This would most likely end the call prematurely so the exception should be thrown only when the Say It Smart plugin cannot do what it is supposed to do and is unable to recover.

Configuration Methods

The following bullets describe the configuration methods:

- `SayItSmartDisplay getDisplayInformation()`

  This method is used to specify display information for Builder for Call Studio to render this plugin's configuration. The `SayItSmartDisplay` object lists the plugin type, input formats, output format, and filesets. Each type, input format, output format, and fileset must have a real name, display name, and description. The relationship between the input formats, output formats, and filesets are defined by the methods listed below.

  This method throws a `SayItSmartException` if the `SayItSmartDisplay` object is not configured correctly.

  `getDisplayInformation()` is abstract, meaning every Say It Smart plugin must implement this method.

- `SayItSmartDependency getFormatDependencies()`

  As described in the design section, a plugin's output formats are dependent on the input formats. This method defines those relationships. The `SayItSmartDependency` object is used to specify which output formats listed in the `getDisplayInformation()` method apply to which input formats.

  The `SayItSmartDependency` class is defined generically with the concepts of *parents* and *children*. For this method, the parents are input formats and the children are output formats. The developer simply uses the `addParent()` methods to add a new input format and the output formats that depend on it. The `addChild()` and `addChildren()` methods are used to add additional output formats that are dependent on the specified parent. This is done until all input formats are listed and all the output formats that depend on it are listed. Remember that the names to use here are the *real names*, the display names are used only by the Builder. All the input formats and output formats defined in the `getDisplayInformation()` method must be mapped here.

  This method throws a `SayItSmartException` if the `SayItSmartDependency` object is not configured correctly.

  `getFormatDependencies()` is abstract, meaning every Say It Smart plugin must implement this method.

- `SayItSmartDependency getFilesetDependencies()`

  This method is identical to the `getFormatDependencies()` method except it defines which filesets are dependent on which output formats. This method also returns a `SayItSmartDependency` object, except this time the *parent* is an output format, and the *child* is a fileset. Again, the *real names* of the output formats and filesets must be used here. All the output formats and filesets defined in the `getDisplayInformation()` method must be mapped here.

  This method throws a `SayItSmartException` if the `SayItSmartDependency` object is not configured correctly.

  `getFilesetDependencies()` is abstract, meaning every Say It Smart plugin must implement this method.

Utility Methods

The following utility methods are provided to aid Say It Smart plugin developers. Their use is optional.

-
  ```
  void validateArguments(Object data, String inputFormat,
                         String outputFormat, String fileset)
  ```

This method validates each argument and throws a `SayItSmartException` if there is an error with one of them. The exception is thrown if `data` is `null`, or `inputFormat`, `outputFormat`, or `fileset` does not correspond to the ones defined in the `getDisplayInformation()` method. Additionally, an exception is thrown if `outputFormat` does not depend on `inputFormat` or `fileset` does not depend on the `outputFormat`. The error message lists all the appropriate options available.

All Unified CVP Say It Smart plugins call this method in the first line of their methods to run.

- `String createError(String header, String[] options, String footer)`

  This utility method is a shortcut for creating an error message when some value does not match one of an array of different possible values. The error conveys that and lists all the values the data can be. Many Unified CVP Say It Smart plugins provide an array of `String` elements containing formats that they will accept input data to arrive in and this method is used when the input data does not arrive in a supported format. The output string starts with *SayItSmart Error – PLUGIN* where PLUGIN is the name of the plugin type as returned by the `getDsplayInformation()` method. The header then appears, followed by a comma-delimited list of correct values (with the last option following an *and*) followed by the footer.

  This is best explained with an example. A Say It Smart plugin named *Foo* has the input formats *a*, *b*, *c*, and *d*. If the input format passed was *e*, this method can be called to create the error message. Calling the message like so:

  ```
    createError("The only input formats supported are: ", new String[] {"a", "b", "c",
  "d"}, " Please use a supported format.").
  ```

  will produce the following error message:

  ```
  SayItSmart Error - Foo: The only input formats supported are: "a", "b", "c", and "d".
  Please use a supported format.
  ```

- `String[] getFilesetContents(String fileset)`

  This utility method returns an array of Strings containing the filenames required for the fileset passed as input. This method can be used to determine what audio files need to be recorded to fully support a fileset for a Say It Smart plugin. At this point, this method exists only for informational purposes, it is not accessed by either Builder for Call Studio or VXML Server.