Say It Smart Specifications for Cisco Unified CVP VXML Server and Cisco Unified Call Studio

Release 4.1(1)

November 2007

Corporate Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 526-4100

Table Of Contents

Preface

Purpose

This document provides specifications for the Say It Smart plug-ins included with Cisco Unified CVP VoiceXML Server.

Audience

This document is intended for voice application developers using Cisco Unified CVP VXML Server and Cisco Unified Call Studio.

Organization

Chapter 1, "Introduction"

Introduces Say It Smart plug-ins.

Chapter 2, "Credit Card"

Describes the Credit Card Say It Smart plug-in.

Chapter 3, "Currency"

Describes the Currency Say It Smart plug-in.

Chapter 4, "Custom Content"

Describes the Custom Content Say It Smart plug-in.

Chapter 5, "Date"

Describes the Date Say It Smart plug-in.

Chapter 6, "Digits"

Describes the Digits Say It Smart plug-in.

Chapter 7, "Filename"

Describes the Filename Say It Smart plug-in.

Chapter 8, "Number"

Describes the Number Say It Smart plug-in.

Chapter 9, "Phone"

Describes the Phone Say It Smart plug-in.

Chapter 10, "Social Security"

Describes the Social Security Say It Smart plug-in.

Chapter 11, "String"

Describes the String Say It Smart plug-in.

Chapter 12, "State"

Describes the State Say It Smart plug-in.

Chapter 13, "Time"

Describes the Time Say It Smart plug-in.

Obtaining Documentation, Obtaining Support, and Security Guidelines

For information on obtaining documentation, obtaining support, providing documentation feedback, security guidelines, and also recommended aliases and general Cisco documents, see the monthly *What's New in Cisco Product Documentation*, which also lists all new and revised Cisco technical documentation, at:

http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html

Related Documentation

- *Cisco Security Agent Installation/Deployment for Cisco Unified Customer Voice Portal* provides installation instructions and information about Cisco Security Agent for the Unified CVP deployment. **We strongly urge you to read this document in its entirety.**
- *Configuration and Administration Guide for Cisco Unified Customer Voice Portal* describes how to set up, run, and administer the Cisco Unified CVP product, including associated configuration.
- *Element Specifications for Cisco Unified CVP VXML Server and Cisco Unified Call Studio* describes the settings, element data, exit states, and configuration options for Elements.
- *Installation and Upgrade Guide for Cisco Unified Customer Voice Portal* describes how to install Unified CVP software, perform initial configuration, and upgrade.
- *Operations Console Online Help for Cisco Unified Customer Voice Portal* describes how to use the Operations Console to configure Unified CVP solution components.
- *Planning Guide for Cisco Unified Customer Voice Portal* provides a product overview and describes how to plan for a Unified CVP deployment.
- *Port Utilization Guide for Cisco Unified Customer Voice Portal* describes the ports used in a Unified CVP deployment.
- *Programming Guide for Cisco Unified CVP VXML Server and Cisco Unified Call Studio* describes how to build components that run on the Cisco Unified CVP VXML Server.
- *Reporting Guide for Cisco Unified Customer Voice Portal* describes the Reporting Server, including how to configure and manage it, and discusses the hosted database.
- *Troubleshooting Guide for Cisco Unified Customer Voice Portal* describes how to isolate and solve problems in the Unified CVP solution.
- *User Guide for Cisco Unified CVP VXML Server and Cisco Unified Call Studio* describes the functionality of Cisco Unified Call Studio including creating projects, using the Cisco Unified Call Studio environment, and deploying applications to the Cisco Unified CVP VXML Server.

Conventions

This manual uses the following conventions:

Convention	Description
boldface font	Boldface font is used to indicate commands, such as user entries, keys, buttons, and folder and submenu names. For example: ▪ Choose **Edit > Find**. ▪ Click **Finish**.
italic font	*Italic* font is used to indicate the following: ▪ To introduce a new term. Example: A *skill group* is a collection of agents who share similar skills. ▪ For emphasis. Example: *Do not* use the numerical naming convention. ▪ A syntax value that the user must replace. Example: IF (*condition, true-value, false-value*) ▪ A book title. Example: See the *Cisco CRS Installation Guide*.

`window font`	`Window font`, such as Courier, is used for the following:
	▪ Text as it appears in code or that the window displays. Example:
	`<html><title>Cisco Systems,Inc. </title></html>`
	▪ File names. Example: `tserver.properties`.
	▪ Directory paths. Example:
	`C:\Program Files\Adobe`

< >	Angle brackets are used to indicate the following: ▪ For arguments where the context does not allow italic, such as ASCII output. ▪ A character string that the user enters but that does not appear on the window such as a password.

Chapter 1: Introduction

Say It Smart is a Unified CVP technology that handles the breakdown of formatted data into an array of audio files played one after the other to render the data in a manner understandable by a caller. While many Text To Speech (TTS) engines can perform a similar function, the power of Say It Smart is that it can handle the playback using pre-recorded audio. Each Say It Smart type lists the audio files required to fully render all the formatted data it can handle. The user need only record these files according to the guidelines specified below and Say It Smart does the rest.

Each Say It Smart type is handled by a separate plug-in deployed on Cisco Unified Call Studio (Call Studio) and Cisco Unified CVP VXML Server (VXML Server). Unified CVP includes many common types such as dates and times. Developers can produce their own plug-ins to either extend Unified CVP Say it Smart plug-in functionality, or introduce new types.

The following defines the characteristics a Say It Smart plug-in requires:

- **Type**. A Say It Smart plug-in is associated with a single type that defines on a high level what kind of data can be handled by the plug-in. Numbers, dates, or currency values are examples of types.
- **Input Format**. A Say It Smart plug-in can have from one to many input formats that define how the data appears when it is sent to the plug-in. These formats may reflect different ways that type can be represented. For example, a date may appear in MMDDYYYY format or YYYYMMDD.
- **Output Format**. A Say It Smart plug-in can have from one to many output formats that define how to express the data passed to the plug-in. Output formats are dependent on input formats, once an input format is changed, the output formats available also change. Output formats can encapsulate differences in expression, such as reading back a value with pauses. They can also reflect language differences or even preferences in how to tailor the output. For example, a time may have an output format that reads 12:00 as “noon” or another that reads back the time in Spanish.
- **Fileset**. A Say It Smart plug-in can have from one to many filesets that list all the audio files required to render a particular output format. Filesets are dependent on output format, once an output format changes, the filesets available also change. Different filesets represent different combinations of files that will render the same data in the specified output format. The most common use of filesets is to use different groups of files to render the data so it sounds better by using more files, or using fewer files but with a more robotic sound. Another use for filesets would be to provide a different gender or playback speed. For example, a fileset may be introduced that reads back a number slowly for those applications where the audience requires it.

- **Audio Files**. Say It Smart plug-ins return a list of audio files needed to render the data in the manner specified by the above criteria. The application designer is required to record all the audio files specified by the fileset(s) they intend on using, name the audio files appropriately, and place them in a centrally servable location. Some criteria on audio files are:
  - All audio files must be given names listed in the specification (with the appropriate audio type extension). All Unified CVP Say It Smart plug-ins use filenames in lowercase and are named such that they can exist on any computing platform without naming issues (the names do not include spaces or unusual punctuation). Any naming inconsistencies will cause Unified CVP Say It Smart plug-ins to use TTS for those files.
  - All audio files for a Say It Smart format must be of a single audio type. Mixing WAV and VOX files, for example, is not possible.
  - Not all files listed need to be recorded. If the user is fairly sure some files will never be encountered, they can be left off. Unified CVP Say It Smart plug-ins use TTS as a backup so if a missing audio file is requested, it will be read as TTS. This may be a bit disconcerting to the caller but does not cause any issues for the application. For example, the Unified CVP Number Say It Smart plug-in can handle numbers up to 999 trillion and the user may know that their application will not handle numbers larger than ten thousand so may choose not to record “million”, “billion”, or “trillion”.
  - Many of the Unified CVP Say It Smart plug-ins use filesets whose contents include audio files specified by the Unified CVP Number Say It Smart plug-in. Recording the audio files to support Number will greatly reduce the number of files needed for other types.
  - All audio files for a particular plug-in must be stored within the same directory. Unified CVP Say It Smart plug-ins require the audio files used by the plug-in to reside in a single directory, though custom plug-ins can require subdirectories of this root directory.
  - Audio files must be placed in a location made accessible via an HTTP request from the voice browser. Unlike the Unified CVP software itself, serving audio files does not require an application server, they can be served by any web server such as IIS or Apache.

Note that for types, input formats, output formats, and filesets, a plug-in defines a name for each as well as a display name. The display name is used for readability purposes and is what Call Studio shows when a new Say It Smart audio item is configured. The actual name is used by VXML Server and the developer when they build dynamic voice element configurations.

The Say It Smart plug-ins requiring the use of a pause produce VoiceXML using the <break> tag. Some voice browsers do not support this tag so Say It Smart playback normally including pauses on these browsers would hear no pauses.

This document presents full specifications for all Unified CVP Say It Smart plug-in types, including all input formats, output formats, filesets, and audio files required. The display names of these are also provided.

Chapter 2: Credit Card

Name: `creditCard`
Display Name: Credit Card
Class Name: `com.audium.sayitsmart.plug-ins.AudiumSayItSmartCreditCard`

Description

This Say It Smart type handles the reading of a credit card number. Any 13, 14, 15, or 16 digit number will be handled. Many times, a credit card number may appear with dashes at certain places in the number. To avoid having to process the data before it is sent to the plug-in, it will understand the credit card number with these optional dashes, though no punctuation other than dashes is allowed. The plug-in reads the credit card number back digit-by-digit, inserting 150 millisecond pauses at certain places where the credit card number is normally divided.

The plug-in Java class can easily be extended to create, in just a few lines of code, a new plug-in performing the same function with a different pause length or additional formatting options.

Input Formats

Name (Display Name)	Description
`cc_number` (13/14/15/16 Digit Number)	The data can be handled in any of the following formats: 16-digit cards (Visa, Mastercard, etc.): ################, ####-####-####-#### 15-digit cards (American Express): ###############, ####-######-##### 14-digit cards (Diner's Club): ##############, ####-#######-#### 13-digit cards (Visa): #############, ####-###-###-###

Output Formats

Name (Display Name)	Input Format Depends On	Description
`digits_with_pauses` (As digits w/ pauses)	`cc_number`	The credit card number is played back digit-by-digit with 150 millisecond pauses where the number is normally divided.

Filesets

Name (Display Name)	Output Format Depends On	Description
`standard` (Standard (0-9))	`digits_with_pauses`	This fileset contains ten files: 0 through 9. It is the only fileset required.

Audio Files

All audio files must be named as appears below. The names do not have an extension, the developer can choose whatever file type supported by their voice browser.

0	1	2	3	4	5	6	7	8	9

Examples

Data:	1234-5678-9012-3456
Input Format:	`cc_number`
Output Format:	`digits_with_pauses`
Fileset:	`standard`
Playback:	“1” “2” “3” “4” <150ms pause> “5” “6” “7” “8” <150ms pause> “9” “0” “1” “2” <150ms pause> “3” “4” “5” “6”

Data:	111122222233333
Input Format:	`cc_number`
Output Format:	`digits_with_pauses`
Fileset:	`standard`
Playback:	“1” “1” “1” “1” <150ms pause> “2” “2” “2” “2” “2” “2” <150ms pause> “3” “3” “3” “3” “3”

Chapter 3: Currency

Name: currency
Display Name: Currency ($)
Class Name: `com.audium.sayitsmart.plug-ins.AudiumSayItSmartCurrency`

Description

This Say It Smart type handles the reading of a currency value in dollars and cents. It only handles dollars and cents, so will work with U.S., Canadian, and other currencies using dollars and cents. The input data can optionally include a dollar sign ($) and does not need to include a decimal point. The amount can be positive or negative, and can even contain an exponent. The amount can be up to $999 trillion. The value is read normally, though if one component is zero, it will not be read. If the decimal contains more than two significant digits it will be rounded to the nearest cent.

This plug-in uses the Unified CVP Number Say it Smart plug-in to render the dollar and cent amounts. It uses the same audio files so if recording was done to support Number, those files can be leveraged to support Currency.

Input Formats

Name (Display Name)	Description
`standard` (Standard Currency)	The data can appear as a standard number with or without a minus sign, decimal point, $ sign, and even an exponent. No commas are allowed.

Output Formats

Name (Display Name)	Input Format Depends On	Description
`dollars_cents` (X dollars and Y cents)	`standard`	The dollar and cent amounts are read separately.

Filesets

Name (Display Name)	Output Format Depends On	Description
`standard` (Standard)	`dollars_cents`	This fileset involves fewer audio files to render the currency amount but at the cost of sounding a bit robotic. This directly correlates to the Unified CVP Number Say it Smart plug-in's `standard` fileset.
`enhanced` (Enhanced)	`dollars_cents`	This fileset involves more audio files to render a better sounding currency amount. This directly correlates to the Unified CVP Number Say It Smart plug-in's `enhanced` fileset.

Audio Files

All audio files must be named as appears below. The names do not have an extension, the developer can choose whatever file type supported by their voice browser.

Standard Fileset

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	30	40	50	60	70	80	90	negative	
hundred	thousand	million	billion	trillion	dollars	dollar	and	cents	cent

Enhanced Fileset

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99
100	200	300	400	500	600	700	800	900	
1000	2000	3000	4000	5000	6000	7000	8000	9000	
negative	thousand	million	billion	trillion	dollars	dollar	and	cents	cent

Examples

Data:	$25052.085
Input Format:	`standard`
Output Format:	`dollars_cents`
Fileset:	`enhanced`
Playback:	“25” “thousand” “52” “dollars” “and” “9” “cents”
Data:	0.01
Input Format:	`standard`
Output Format:	`dollars_cents`
Fileset:	`standard`
Playback:	“1” “cent”

Data:	6.99E4
Input Format:	`standard`
Output Format:	`dollars_cents`
Fileset:	`standard`
Playback:	“60” “9” “thousand” “9” “hundred” “dollars”
Data:	-$69900
Input Format:	`standard`
Output Format:	`dollars_cents`
Fileset:	`enhanced`
Playback:	“negative” “69” “thousand” “900” “dollars”

Chapter 4: Custom Content

Name: literal
Display Name: Custom Content
Class Name: `com.audium.sayitsmart.plug-ins.AudiumSayItSmartLiteral`

Description

This Say It Smart type was introduced to provide several helpful and time saving features to the application designer and developer:

- Provide a way to allow a list of audio files (with TTS transcripts) of variable length to be played one after the other in one audio item.
- Provide a more direct link to internal Java classes that may contain dynamic audio content as an alternative to creating dynamic voice element configurations.
- Provide at least the same functionality as the now "deprecated" File and String Unified CVP Say It Smart types.

Input Formats

Name (Display Name)	**Description**
`simple` (String (No Delimiters))	A text string that can represent a single filename or a single TTS string.
`complex` (FILE:::TTS|||… |||FILE:TTS)	A text string that follows a specific format with delimiters in order to represent any number of audio files and TTS transcripts. An audio file is separated from its TTS transcript by three colons. Each audio file/TTS combination is separated from others by three pipes. Note that each component of the combination can be blank if no audio file or TTS content is necessary. The audio will be played in the order in which it appears in the string from left to right.
`resultset` (ResultSetList Object)	A Java `ResultSetList` object that has been created by the Unified CVP Database element as a result of a database query that is expected to contain audio information. The result must return two columns, the first being the audio file (or `null` if no audio file is needed) and the second column being the TTS transcript for the audio file (or `null` if there is no TTS transcript). There can be any number of rows. The audio will be played in the order in which it appears in the result set.

Name (Display Name)	Description
`siscontent` (SayItSmartContent Object)	Each Say It Smart plug-in's Java code creates a `SayItSmartContent` object to represent audio content that is then passed to Unified CVP VXML Server to render into VoiceXML. This input format accepts a developer-created object of this type and the plug-in will pass this to VXML Server without making any modifications. This object can contain any number of audio files, TTS transcripts, and pauses the developer desires.
`array` (String[] Object)	A `String` array that can contain either a list of audio filenames or TTS transcripts (it cannot contain a mixture of audio filenames and TTS transcripts). The audio will be played in the order it appears in the array.

Output Formats

Name (Display Name)	Input Format Depends On	Description
`standard` (Filename w/ TTS Backup)	`complex resultset siscontent`	This output format will produce output containing both audio files (if defined) and TTS transcripts (if defined), assuming that the TTS content may contain Speech Synthesis Markup Language (SSML). This adds some additional overhead so use the `standard_no_ssml` output format if it is known that the TTS transcripts do not contain SSML.
`standard_no_ssml` (Filename w/ TTS Backup (no SSML))	`complex resultset siscontent`	This output format will produce output containing both audio files (if defined) and TTS transcripts (if defined), assuming that the TTS content does not contain SSML. Assuming no SSML makes the process more efficient than keeping open the possibility that the TTS content may have SSML (as in the `standard` fileset).
`tts` (TTS Only)	`simple complex resultset siscontent array`	This output format will produce output containing only the TTS content of the data, even if it contains audio file content. For the `simple` and `array` input formats, this output format indicates that the data contains only TTS content. This output format assumes the TTS content may contains SSML. This adds some additional overhead so use the `tts_no_ssml` output format if it is known that the TTS content does not contain SSML.
`tts_no_ssml` (TTS Only (no SSML))	`simple complex resultset siscontent array`	This output format will produce output containing only the TTS content of the data, even if it contains audio file content. For the `simple` and `array` input formats, this output format indicates that the data contains only TTS content. Assuming no SSML makes the process more efficient than keeping open the possibility that the TTS content may have SSML (as in the `tts` fileset).

Name (Display Name)	Input Format Depends On	Description
`files` (Filename(s) Only)	`simple` `complex` `resultset` `siscontent` `array`	This output format will produce output containing only the audio file content of the data, even if it contains TTS content. For the `simple` and `array` input formats, this output format indicates that the data contains audio files only.

Filesets

Name (Display Name)	Output Format Depends On	Description
`none` (No Fileset)	`standard` `standard_no_ssml` `tts` `tts_no_ssml` `files`	This plug-in allows the developer to specify any amount of audio files, the names of which are determined at runtime. As a result, there is no need for a fileset. Every Say It Smart plug-in, though, requires at least one fileset, so this one is simply named "none".

Audio Files

None. The audio files will be determined by the application designer and developer.

Examples

Data:	myGreeting.wav
Input Format:	`simple`
Output Format:	`files`
Fileset:	`none`
Playback:	myGreeting.wav (with no TTS backup)

Data:	This is some text to speech
Input Format:	`simple`
Output Format:	`tts_no_ssml`
Fileset:	`none`
Playback:	"This is some text to speech" (this is read as TTS)

Data:	a.wav:::backup for a|||b.wav:::backup for b
Input Format:	`complex`
Output Format:	`standard_no_ssml`
Fileset:	`none`
Playback:	a.wav (with TTS backup "backup for a") b.wav (with TTS backup "backup for b")

Data:	a.wav:::|||:::some <break size="large"> tts
Input Format:	`complex`
Output Format:	`standard`
Fileset:	`none`
Playback:	a.wav (with no TTS backup) "some " <large pause> " tts" (no audio file played, SSML tags included in VoiceXML)

There are no examples of input formats that take Java objects as the data must be created by a developer in custom Java code.

Chapter 5: Date

Name: date
Display Name: Date
Class Name: `com.audium.sayitsmart.plug-ins.AudiumSayItSmartDate`

Description

This Say It Smart type handles the reading of a date or portions of a date. It handles many input formats for the date, some of which provide only a partial date. The plug-in also supports the components of the date separated by forward slashes (/) and will require the use of this delimiter if any component of the date is expressed with one digit instead of two (for example, May 2 can be expressed as 0502 or 5/2 where the slash is required if any component is not padded with 0s). The date is read back in standard English fashion; the month name (rather than the number), the day, and the year. If only partial information is available, only that data will be read. The plug-in will only read legitimate dates according to the standard Gregorian calendar and will throw an error if an incorrect date is given.

This plug-in uses the Unified CVP Number Say it Smart plug-in to render the year. It uses the same audio files so recordings done to support Number can be leveraged to support Date.

Input Formats

All input formats with more than one date component can appear delimited with forward slashes.

Name (Display Name)	Description
`mmddyyyy` (MMDDYYYY)	The full date with the month, day, and four digit year. The data can be handled in any of the following formats: `mmddyyyy`, `mm/dd/yyyy`, `m/dd/yyyy`, `mm/d/yyyy`, or `m/d/yyyy`.
`mmddyy` (MMDDYY)	The full date with the month, day, and two digit year. The data can be handled in any of the following formats: `mmddyy`, `mm/dd/yy`, `m/dd/yy`, `mm/d/yy`, `mm/dd/y`, `m/d/yy`, `m/dd/y`, `mm/d/y`, or `m/d/y`.
`ddmmyyyy` (DDMMYYYY)	The full date with the day, month, and four digit year. The data can be handled in any of the following formats: `ddmmyyyy`, `dd/mm/yyyy`, `d/mm/yyyy`, `dd/m/yyyy`, or `d/m/yyyy`.
`ddmmyy` (DDMMYY)	The full date with the day, month, and two digit year. The data can be handled in any of the following formats: `ddmmyy`, `dd/mm/yy`, `d/mm/yy`, `dd/m/yy`, `dd/mm/y`, `d/m/yy`, `d/mm/y`, `dd/m/y`, or `d/m/y`.
`yyyymmdd` (YYYYMMDD)	The full date with the four digit year, month, and day. The data can be handled in any of the following formats: `yyyymmdd`, `yyyy/mm/dd`, `yyyy/m/dd`, `yyyy/mm/d`, or `yyyy/m/d`.
`mmyyyy` (MMYYYY)	The month and four digit year. The data can be handled in any of the following formats: `mmyyyy`, `mm/yyyy`, or `m/yyyy`.

Name (Display Name)	Description
mmyy (MMYY)	The month and two digit year. The data can be handled in any of the following formats: mmyy, mm/yy, m/yy, mm/y, or m/y.
mmdd (MMDD)	The month and day. The data can be handled in any of the following formats: mmdd, mm/dd, m/dd, mm/d, or m/d.
yyyy (YYYY)	The four digit year alone. The data can be handled in any of the following formats: yyyy.
ddmm (DDMM)	The day and month. The data can be handled in any of the following formats: ddmm, dd/mm, d/mm, dd/m, or d/m.
mm (MM)	The month alone. The data can be handled in any of the following formats: mm.

Output Formats

Name (Display Name)	Input Format Depends On	Description
date (The Date)	mmddyyyy ddmmyyyy yyyymmdd	For all input formats containing the full date, this output format plays the month name, day, and full four digit year.
date_19 (The Date w/ YY=19)	mmddyy ddmmyy	For all input formats containing the full date and a two digit year, this plays the month name, day, and year assuming it is in the 1900s.
date_20 (The Date w/ YY=20)	mmddyy ddmmyy	For all input formats containing the full date and a two digit year, this plays the month name, day, and year assuming it is in the 2000s.
month_year (Month/Year)	mmyyyy	Plays the month name and full four digit year.
month_year_19 (Month/Year w/ YY=19)	mmyy	Plays the month name and year assuming it is in the 1900s.
month_year_20 (Month/Year w/ YY=20)	mmyy	Plays the month name and year assuming it is in the 2000s.
month_day (Month/Day)	mmdd ddmm	Plays the month name and the day.
month (Month)	mm	Plays the month name only.
year (Year)	yyyy	Plays the full four digit year only.

Filesets

Name (Display Name)	Output Format Depends On	Description
standard_date (Standard Full Date)	date date_19 date_20	This fileset contains all files needed to render the full date. It involves fewer audio files to render the year but at the cost of sounding a bit robotic. This directly correlates to the Unified CVP Number Say it Smart plug-in's standard fileset.
enhanced_date (Enhanced Full Date)	date date_19 date_20	This fileset contains all files needed to render the full date. This fileset involves more audio files to render a better sounding year. This directly correlates to the Unified CVP Number Say It Smart plug-in's enhanced fileset.
month_standard_year (Month/Standard Year)	month_year month_year_19 month_year_20	This fileset contains all files needed to render a month and a year. It involves fewer audio files to render the year but at the cost of sounding a bit robotic. This directly correlates to the Unified CVP Number Say it Smart plug-in's standard fileset.
month_enhanced_year (Month/Enhanced Year)	month_year month_year_19 month_year_20	This fileset contains all files needed to render a month and a year. This fileset involves more audio files to render a better sounding year. This directly correlates to the Unified CVP Number Say It Smart plug-in's enhanced fileset.
month_day (Month/Day)	month_day	This fileset contains all files needed to render a month and a day.
month (Month Only)	month	This fileset contains all files needed to render the month alone.
standard_year (Standard Year)	year	This fileset contains all files needed to render the year alone. It involves fewer audio files but at the cost of sounding a bit robotic. This directly correlates to the Unified CVP Number Say it Smart plug-in's standard fileset.
enhanced_year (Enhanced Year)	year	This fileset contains all files needed to render the year alone. This fileset involves more audio files to render a better sounding year. This directly correlates to the Unified CVP Number Say It Smart plug-in's enhanced fileset.

Audio Files

All filesets including the month have a separate file for each month. All filesets with the day of the month will have a separate file for each day ("1st", "2nd", etc). Only those filesets containing the year have standard and enhanced versions that render the year with less files or more files respectively. The files required to render the year are almost the same as the Unified CVP Number Say it Smart plug-in with the exception that numbers greater than 9999 are not necessary and zero is replaced with "oh".

Standard Full Date

january	february	march	april	may	june	july	august	september	october
november	december								
1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
11th	12th	13th	14th	15th	16th	17th	18th	19th	20th
21st	22nd	23rd	24th	25th	26th	27th	28th	29th	30th
31st									
oh	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	30	40	50	60	70	80	90	hundred	thousand

Enhanced Full Date

january	february	march	april	may	june	july	august	september	october
november	december								
1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
11th	12th	13th	14th	15th	16th	17th	18th	19th	20th
21st	22nd	23rd	24th	25th	26th	27th	28th	29th	30th
31st									
oh	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99
100	200	300	400	500	600	700	800	900	
1000	2000	3000	4000	5000	6000	7000	8000	9000	hundred

Month/Standard Year

january	february	march	april	may	june	july	august	september	october
november	december								
oh	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	30	40	50	60	70	80	90	hundred	thousand

Month/Enhanced Year

january	february	march	april	may	june	july	august	september	october
november	december								
oh	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99
100	200	300	400	500	600	700	800	900	
1000	2000	3000	4000	5000	6000	7000	8000	9000	hundred

Month/Day

january	february	march	april	may	june	july	august	september	october
november	december								
1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
11th	12th	13th	14th	15th	16th	17th	18th	19th	20th
21st	22nd	23rd	24th	25th	26th	27th	28th	29th	30th
31st									

Month Only

january	february	march	april	may	june	july	august	september	october
november	december								

Standard Year

oh	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	30	40	50	60	70	80	90	hundred	thousand

Enhanced Year

oh	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99
100	200	300	400	500	600	700	800	900	
1000	2000	3000	4000	5000	6000	7000	8000	9000	hundred

Examples

Data:	02171971
Input Format:	`mmddyyyy`
Output Format:	`date`
Fileset:	`standard_date`
Playback:	“february” “17th” “19” “70” “1”

Data:	02/09/05
Input Format:	`ddmmyy`
Output Format:	`date_19`
Fileset:	`enhanced_date`
Playback:	“september” “2nd” “19” “oh” “5”

Data:	072003
Input Format:	`mmyyyy`
Output Format:	`month_year`
Fileset:	`month_standard_year`
Playback:	“july” “2” “thousand” “3”

Data:	2387
Input Format:	`yyyy`
Output Format:	`year`
Fileset:	`enhanced_year`
Playback:	“23” “87”

Data:	12
Input Format:	`mm`
Output Format:	`month`
Fileset:	`month`
Playback:	“december”

Data:	10/10
Input Format:	`mmdd`
Output Format:	`month_day`
Fileset:	`month_day`
Playback:	“october” “10th”

Chapter 6: Digits

Name: digits
Display Name: Digit-By-Digit
Class Name: `com.audium.sayitsmart.plug-ins.AudiumSayItSmartDigit`

Description

This Say It Smart type handles the reading of any number digit by digit. The number can be negative or positive and can also contain a decimal (though, unlike Number, exponents are not supported). Every character is read individually.

Input Formats

Name (Display Name)	Description
`number` (Any Length Number)	This number can appear as any length whole or decimal number. If the number is negative, the minus sign must be the first character.

Output Formats

Name (Display Name)	Input Format Depends On	Description
`digits` (Digit-By-Digit)	`number`	The number can be played back in only one manner: digit by digit.

Filesets

Name (Display Name)	Output Format Depends On	Description
`standard` (Standard)	`digits`	This single fileset contains all numbers from 0 to 9 as well as “point” and “negative”.

Audio Files

0	1	2	3	4	5	6	7	8	9
negative	point								

Examples

Data:	96.89
Input Format:	`number`
Output Format:	`digits`
Fileset:	`standard`
Playback:	”9” ”6” ”point” ”8” ”9”

Data:	-10
Input Format:	`number`
Output Format:	`digits`
Fileset:	`standard`
Playback:	“negative“ “1” “0”

Chapter 7: Filename

Name: file
Display Name: Filename
Class Name: `com.audium.sayitsmart.plug-ins.AudiumSayItSmart`

Description

This Say It Smart type handles the playback of an audio file whose name is passed as input to the plug-in. In Call Studio, one can specify a file type to apply to all audio files listed by the Say It Smart type. Filename is no different, the file type extension specified in Call Studio will be appended to the filename passed to the plug-in. If the data sent as input already has an extension, Call Studio file type should be blank. For a TTS backup, the plug-in returns the name of the audio file since the transcript cannot be known in advance. When trying to use this type in TTS only mode, it returns a `null`.

Important Note: In Call Studio and VXML Server substitution can be used within audio file names and TTS content, so one can do with substitution what this plug-in does. Additionally, a new Say It Smart plug-in type was introduced: Custom Content, that does what this plug-in does and more (such as allowing for a TTS backup). As a result, this plug-in should be considered "deprecated". It is still included for backwards compatibility however eventually this plug-in will no longer be included in Unified CVP updates and it is recommended to use one of the above solutions instead of using this plug-in.

Input Formats

Name (Display Name)	Description
`string` (A Filename)	Any string (the plug-in does no filename validation).

Output Formats

Name (Display Name)	Input Format Depends On	Description
`audio` (Audio File)	`string`	A single audio file whose name is passed to the plug-in.

Filesets

Name (Display Name)	Output Format Depends On	Description
`none` (No Fileset)	`audio`	The fileset contains only one file: the one to play.

Audio Files

The only audio file needed is the audio file to play, which is determined dynamically.

Examples

Data:	my file
Input Format:	`string`
Output Format:	`audio`
Fileset:	`none`
Playback:	[Assuming an extension of “ulaw” was given in Call Studio] “my file.ulaw”

Data:	audio_logo.wav
Input Format:	`string`
Output Format:	`audio`
Fileset:	`none`
Playback:	[Assuming an extension of ”wav” was given in Call Studio] “audio_logo.wav.wav”

Chapter 8: Number

Name: number
Display Name: Number
Class Name: `com.audium.sayitsmart.plug-ins.AudiumSayItSmartNumber`

Description

This Say It Smart type handles the reading of any number. The number can be negative or positive, contain a decimal, and can even contain an exponent. The whole part of the number is read normally and the decimal part of the number is read digit-by-digit. This plug-in can handle numbers up to 999 trillion.

The number can be read back in a way that sounds somewhat robotic, though it uses a minimum number of audio files. The number can also be read back in a manner that sounds better to the caller but will require more files to do so. These differences are encapsulated in the Number type's two filesets: `standard` and `enhanced`. All Unified CVP Say It Smart plug-ins that have numerical components use the Number plug-in to convert their numbers so those plug-ins will list these two filesets as well.

Input Formats

Name (Display Name)	Description
`standard` (Standard)	This represents any number, negative or positive, with or without a decimal, and optionally containing an exponent. No commas are allowed.

Output Formats

Name (Display Name)	Input Format Depends On	Description
`standard` (Standard Number)	`standard`	The whole part of the number is read normally and the decimal is read digit-by-digit.
`no_trailing_0s` (Read w/ no Trailing 0s)	`standard`	The whole part of the number is read normally, the decimal is read digit-by-digit, omitting trailing zeros.

Filesets

Name (Display Name)	Output Format Depends On	Description
`standard` (Standard)	`standard` `no_trailing_0s`	This fileset involves fewer audio files to render the number but at the cost of sounding a bit robotic.
`enhanced` (Enhanced)	`standard` `no_trailing_0s`	This fileset involves more audio files to render a better sounding number.

Audio Files

Standard Fileset

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	30	40	50	60	70	80	90		
negative	point	hundred	thousand	million	billion	trillion			

Enhanced Fileset

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99
100	200	300	400	500	600	700	800	900	
1000	2000	3000	4000	5000	6000	7000	8000	9000	
negative	point	thousand	million	billion	trillion				

Examples

Data:	4836945.160
Input Format:	`standard`
Output Format:	`standard`
Fileset:	`enhanced`
Playback:	“4” “million” “800” “36” “thousand” “900” “45” “point” “1” “6” “0”

Data:	3.10
Input Format:	`standard`
Output Format:	`no_trailing_0s`
Fileset:	`standard`
Playback:	“3” “point” “1”

Data:	36.1234E2
Input Format:	`standard`
Output Format:	`standard`
Fileset:	`standard`
Playback:	“3” “thousand” “6” “hundred” “12” “point” “3” “4”

Data:	-3E-2
Input Format:	`standard`
Output Format:	`standard`
Fileset:	`standard`
Playback:	“negative” “0” “point” “0” “0” “3”

Chapter 9: Phone

Name: phone
Display Name: Phone Number
Class Name: `com.audium.sayitsmart.plug-ins.AudiumSayItSmartPhone`

Description

This Say It Smart type handles the reading of a 10 digit phone number. The number must have an area code and cannot be an 11-digit number starting with 1. Many times, a phone number may appear with various formatting. To avoid having to process the data before it is sent to the plug-in, the plug-in will understand the standard phone number formats. The phone number is read digit-by-digit, inserting 150 millisecond pauses after the area code and exchange.

The plug-in Java class can easily be extended to create, in just a few lines of code, a new plug-in performing the same function with a different pause length or additional formatting options.

Input Formats

Name (Display Name)	Description
`10_digit_whole_number` (10 Digit Number)	The data can be handled in any of the following formats: ##########, (###) ###-####, (###)###-####, ###-###-####, ###.###.####, (###)#######. Note the second format contains a space after the area code.

Output Formats

Name (Display Name)	Input Format Depends On	Description
`digits_with_pauses` (As Digits w/ Pauses)	`10_digit_whole_number`	The phone number is played back digit-by-digit with 150 millisecond pauses where the number is normally divided.

Filesets

Name (Display Name)	Output Format Depends On	Description
`standard` (Standard (0-9))	`digits_with_pauses`	This fileset contains ten files: 0 through 9. It is the only fileset required.

Audio Files

All audio files must be named as appears below. The names do not have an extension, the developer can choose whatever file type supported by their voice browser.

0	1	2	3	4	5	6	7	8	9

Examples

Data:	(800) 555-1212
Input Format:	`10_digit_whole_number`
Output Format:	`digits_with_pauses`
Fileset:	`standard`
Playback:	“8” “0” “0” <150ms pause> “5” “5” “5” <150ms pause> “1” “2” “1” “2”

Data:	1112223333
Input Format:	`10_digit_whole_number`
Output Format:	`digits_with_pauses`
Fileset:	`standard`
Playback:	“1” “1” “1” <150ms pause> “2” “2” “2” <150ms pause> “3” “3” “3” “3”

Chapter 10: Social Security

Name: ssn
Display Name: Social Security Number
Class Name: `com.audium.sayitsmart.plug-ins.AudiumSayItSmartSocialSecurity`

Description

This Say It Smart type handles the reading of a 9-digit social security number. Many times, a social security number may appear with dashes after the third and fifth digits. To avoid having to process the data before it is sent to the plug-in, it will understand the social security number with these optional dashes, though no punctuation other than dashes is allowed. It reads it back digit-by-digit, inserting 150 millisecond pauses after the third and fifth digits.

The plug-in Java class can easily be extended to create, in just a few lines of code, a new plug-in performing the same function with a different pause length or additional formatting options.

Name (Display Name)	Description
`9_digit_whole_number` (9 Digit Number)	The data can be handled in any of the following formats: #########, ###-##-####

Output Formats

Name (Display Name)	Input Format Depends On	Description
`digits_with_pauses` (As Digits w/ Pauses)	`9_digit_whole_number`	The social security number is played back digit-by-digit with 150 millisecond pauses after the third and fifth digits.

Filesets

Name (Display Name)	Output Format Depends On	Description
`standard` (Standard (0-9))	`digits_with_pauses`	This fileset contains ten files: 0 through 9. It is the only fileset required.

Audio Files

0	1	2	3	4	5	6	7	8	9

Examples

Data:	123-45-6789
Input Format:	`9_digit_whole_number`
Output Format:	`digits_with_pauses`
Fileset:	`standard`
Playback:	“1” ”2” ”3” <150ms pause> ”4” ”5” <150ms pause> ”6” ”7” ”8” ”9”

Data:	111223333
Input Format:	`9_digit_whole_number`
Output Format:	`digits_with_pauses`
Fileset:	`standard`
Playback:	“1” ”1” ”1” <150ms pause> ”2” ”2” <150ms pause> ”3” ”3” ”3” ”3”

Chapter 11: String

Name: string
Display Name: TTS String
Class Name: `com.audium.sayitsmart.plug-ins.AudiumSayItSmartString`

Description

This Say It Smart type plays back the data sent as input in Text To Speech (TTS). Even when the "Use Recorded Audio" checkbox is checked in Call Studio, the output will be a TTS string containing the passed data. The input data is unmodified unless it contains characters not allowed in XML and the TTS content is not contained within CDATA (this occurs only on some supported voice browsers). These characters will then be converted to their escaped equivalents (for example "<" is converted to "<").

Important Note: In Call Studio and VXML Server substitution can be used within audio file names and TTS content, so one can do with substitution what this plug-in does. Additionally, a new Say It Smart plug-in type was introduced: Custom Content, that does what this plug-in does and more. As a result, this plug-in should be considered "deprecated". It is still included for backwards compatibility however eventually this plug-in will no longer be included in Unified CVP updates and it is recommended to use one of the above solutions instead of using this plug-in.

Input Formats

Name (Display Name)	Description
`string` (A String)	Any string. The string is modified only when the string contains characters illegal to XML and the TTS content is not placed inside CDATA.

Output Formats

Name (Display Name)	Input Format Depends On	Description
`tts` (The String in TTS)	`string`	The data will be read by the TTS engine.

Filesets

Name (Display Name)	Output Format Depends On	Description
`none` (No Fileset)	`audio`	There is no fileset because this type will never involve the playing of pre-recorded audio files. Every Say It Smart plug-in, though, requires at least one fileset, so this one is simply named "none".

Audio Files

None. The data will always be rendered in TTS.

Examples

Data:	Today's bingo number is 28.
Input Format:	`string`
Output Format:	`tts`
Fileset:	`none`
Playback:	Today's bingo number is 28 (as TTS).

Data:	myfile.wav
Input Format:	`string`
Output Format:	`tts`
Fileset:	`none`
Playback:	myfile.wav (as TTS).

Chapter 12: State

Name: state
Display Name: U.S./Canada State
Class Name: `com.audium.sayitsmart.plug-ins.AudiumSayItSmartState`

Description

This Say It Smart type handles the reading of a U.S. or Canadian state, territory, or province. The data is passed as the two-letter abbreviation of the state and the plug-in plays back the full name. Please see the Audio Files section to see a list of U.S and Canadian states, territories, and provinces. Note that when the VoiceXML is produced, the TTS transcript will be exactly the same as the audio filename except without any underscores.

Input Formats

Name (Display Name)	Description
`state_abbreviation` (2-Character Abbreviation)	A two letter abbreviation of the state (case insensitive).

Output Formats

Name (Display Name)	Input Format Depends On	Description
`state_name` (Full State Name)	`state_abbreviation`	An audio file playing the full state, territory, or province name.

Filesets

Name (Display Name)	Output Format Depends On	Description
`standard` (Standard)	`state_name`	There is only one fileset: a separate audio file for each U.S. or Canadian state, territory or province.

Audio Files

The filenames are as shown (no spaces in the names). The two-letter abbreviation for each state, territory, or province is listed in parentheses.

U.S. Territories

american_samoa (AS)	federated_states_of_micronesia (FM)	guam (GU)
marshall_islands (MH)	northern_mariana_islands (MP)	puerto_rico (PR)
us_virgin_islands (VI)	palau (PW)	

U.S States

alabama (AL)	alaska (AK)	arizona (AZ)	arkansas (AR)
california (CA)	colorado (CO)	connecticut (CT)	delaware (DE)
district_of_columbia (DC)	florida (FL)	georgia (GA)	hawaii (HI
idaho (ID)	illinois (IL)	indiana (IN)	iowa (IA)
kansas (KS)	kentucky (KY)	louisiana (LA)	maine (ME)
maryland (MD)	massachusetts (MA)	michigan (MI)	minnesota (MN)
mississippi (MS)	missouri (MO)	montana (MT)	nebraska (NE)
nevada (NV)	new_hampshire (NH)	new_jersey (NJ)	new_mexico (NM)
new_york (NY)	north_carolina (NC)	north_dakota (ND)	ohio (OH)
oklahoma (OK)	oregon (OR)	pennsylvania (PA)	rhode_island (RI)
south_carolina (SC)	south_dakota (SD)	tennessee (TN)	texas (TX)
utah (UT)	vermont (VT)	virginia (VA)	washington (WA)
west_virginia (WV)	wisconsin (WI)	wyoming (WY)	

Canadian Provinces / Territories

alberta (AB)	british_columbia (BC)	manitoba (MB)	new_brunswick (NB)
newfoundland (NL)	nova_scotia (NS)	northwest_territories (NT)	nunavut (NU)
ontario (ON)	prince_edward (PE)	quebec (QC)	sasketchewan (SK)
yukon (YT)			

Examples

Data:	nY
Input Format:	`state_abbreviation`
Output Format:	`state_name`
Fileset:	`standard`
Playback:	“new_york”

Data:	SK
Input Format:	`state_abbreviation`
Output Format:	`state_name`
Fileset:	`standard`
Playback:	“sasketchewan”

Chapter 13: Time

Name: time
Display Name: Time/Time Period
Class Name: `com.audium.sayitsmart.plug-ins.AudiumSayItSmartTime`

Description

This Say It Smart type handles the playback of the time or a time period. Whether to play back the time or a time period is specified by an input format. The plug-in also supports the different components of the time separated by colons (:) and will require the use of this delimiter if any component of the time is expressed with one digit instead of two (for example, 1:09 AM can be expressed as 0109 or 1:9 where the colon is required if any component is not padded with 0s). The time arrives in 24-hour military format and time periods arrive in combinations of hours, minutes, and seconds. The time is read back in standard English fashion; the hour, the minute, and either "A.M." or "P.M.". Time periods are read back with each component followed by a qualifier ("hours", "minutes", or "seconds"). The plug-in will only read the time or time period if it is legitimate (the components are within the appropriate range).

This plug-in uses the Unified CVP Number Say it Smart plug-in to render each component of the time or time period. It uses the same audio files so recordings done to support Number can be leveraged to support Time.

Input Formats

Name (Display Name)	**Description**
`time_hhmm` (24Hr Time (HHMM))	This input format is used to specify the time. It must arrive in 24-hour format with the hours from 00 to 23 and the minute from 00 to 59. The data can be handled in any of the following formats: `hhmm`, `hh:mm`, `h:mm`, `hh:m`, or `h:m`.
`period_hhmmss` (Time Period (HHMMSS))	This input format is used to specify a time period including hours (from 00 to 99), minutes (from 00 to 59), and seconds (from 00 to 59). The data can be handled in any of the following formats: `hhmmss` or `hh:mm:ss`.
`period_hhmm` (Time Period (HHMM))	This input format is used to specify a time period including hours (from 00 to 99) and minutes (from 00 to 59). The data can be handled in any of the following formats: `hhmm` or `hh:mm`.
`period_mmss` (Time Period (MMSS))	This input format is used to specify a time period including minutes (from 00 to 99) and seconds (from 00 to 59). The data can be handled in any of the following formats: `mmss` or `mm:ss`.

Output Formats

Name (Display Name)	Input Format Depends On	Description
time (The Time)	time_hhmm	The time is read back with the hour (from 1 to 12) followed by the minute (from 0 to 59) followed by "A.M." or "P.M.". If the minute is zero, it will be omitted.
time_special_12 (The Time 12=Midnight/Noon)	time_hhmm	The time is read back exactly as above except that 00:00 is read as "midnight" and 12:00 is read as "noon".
period (Time Period)	period_hhmmss period_hhmm period_mmss	The time period is read back with each component followed by the qualifier "hours", "minutes", or "seconds". If one component is zero, it is omitted.

Filesets

Name (Display Name)	Output Format Depends On	Description
standard_time (Standard Time)	time	This fileset involves fewer audio files to render the time but at the cost of sounding a bit robotic. This directly correlates to the Unified CVP Number Say it Smart plug-in's standard fileset.
enhanced_time (Enhanced Time)	time	This fileset involves more audio files to render a better sounding time. This directly correlates to the Unified CVP Number Say It Smart plug-in's enhanced fileset.
standard_special_12 (Standard Time + Noon/Midnight)	time_special_12	This fileset is exactly the same as standard_time except with two extra files; "noon" and "midnight".
enhanced_special_12 (Enhanced Time + Noon/Midnight)	time_special_12	This fileset is exactly the same as enhanced_time except with two extra files; "noon" and "midnight".
standard_period (Standard Time Period)	period	This fileset involves fewer audio files to render the time period but at the cost of sounding a bit robotic. This directly correlates to the Unified CVP Number Say it Smart plug-in's standard fileset.
enhanced_period (Enhanced Time Period)	period	This fileset involves more audio files to render a better sounding time period. This directly correlates to the Unified CVP Number Say It Smart plug-in's enhanced fileset.

Audio Files

Note that when reading back a time, zeros are replaced by “oh”. for example, 13:05 is read back as “one oh five P.M.”. This is not the case for time periods.

standard_time

oh	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	30	40	50	am	pm				

enhanced_time

oh	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
am	pm								

standard_special_12

oh	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	30	40	50	am	pm	noon	midnight		

enhanced_special_12

oh	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
am	pm	noon	midnight						

standard_period

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	30	40	50	60	70	80	90		
hour	hours	minute	minutes	second	seconds				

enhanced_period

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49
50	51	52	53	54	55	56	57	58	59
60	61	62	63	64	65	66	67	68	69
70	71	72	73	74	75	76	77	78	79
80	81	82	83	84	85	86	87	88	89
90	91	92	93	94	95	96	97	98	99
hour	hours	minute	minutes	second	seconds				

Examples

Data:	20:43
Input Format:	`time_hhmm`
Output Format:	`time`
Fileset:	`standard_time`
Playback:	“8” “40” “3” “pm”

Data:	20:43
Input Format:	`time_hhmm`
Output Format:	`time`
Fileset:	`enhanced_time`
Playback:	“8” “43” “pm”

Data:	0000
Input Format:	`time_hhmm`
Output Format:	`time_special_12`
Fileset:	`standard_special_12`
Playback:	“midnight”

Data:	02:00
Input Format:	`time_hhmm`
Output Format:	`time_special_12`
Fileset:	`enhanced_special_12`
Playback:	“2” “am”

Data:	12:09
Input Format:	`time_hhmm`
Output Format:	`time`
Fileset:	`standard_time`
Playback:	“12” “oh” “9” “pm”

Data:	810001
Input Format:	`period_hhmmss`
Output Format:	`period`
Fileset:	`standard_period`
Playback:	“80” “1” “hours” “1” “second”

Data:	0001
Input Format:	`period_hhmm`
Output Format:	`period`
Fileset:	`standard_period`
Playback:	“1” “minute”

Data:	99:59
Input Format:	`period_mmss`
Output Format:	`period`
Fileset:	`enhanced_period`
Playback:	“99” “minutes” “59” “seconds”

Index