Python Macro Language for Dragon NaturallySpeaking

(c) Copyright 1999 by Joel Gould

Portions (c) Copyright 1999 by Dragon Systems, Inc.

natlink.txt

Documentation for NatLink which connects Python to NatSpeak

April 1, 2000

- added GramObj. setSelectText,getSelectText

- added ResObj. getSelectInfo

- added setTrayIcon

March 24, 2000

- added getWordInfo, getWordProns

- addWord now supports pronunciations

October 21, 1999

- added deleteWord, addWord and setWordInfo

September 22, 1999

- added GramObj.setContext

- added exception WrongType

September 17, 1999

- added extra parameter to inputFromFile

- fix bug which prevents us from loading raw dictation grammars

September 13, 1999

- added hypothesis parameter to GramObj.load() and

GramObj.setHypothesisCallback()

September 9, 1999

- added getWordInfo

September 3, 1999

- added new sample program called windict.py which demonstrates the

use of the DictObj for dictation

- fixed a few bugs in Python threading when using DictObj which

windict exposed.

August 26, 1999

- added new file called wavtools.py which contains miscelleanous

routines for dealing with wave files and word names. Modified

trainuser.py to use this new file.

- the Natlink output window now resets its contents when it is

closed.

- fixed another error thrown by getCurrentModule, this case when you

try to talk to a system modal dialog box.

August 18, 1999

- added a check for a NULL window handle in getCurrentModule which should

fix a reported SRERR\_INVALIDPARAM error occassionally thrown from this

function.

August 12, 1999

- added code which is confitionally included only when an INHOUSE flag is

set; the inhouse-only code is not inlcuded in the standard release

- fixed a few cases of screwing up Python reference counts

August 10, 1999

- added DictObj including the new source code files (dictobj.\*)

- added more tests to testnatlink.py

- fixed a few minor bugs:

= memory leak when recognitionMimic failed which prevented shutdown

= wrong exception raised when bad window handle passed to activate

July 23, 1999

- converted tabs to spaces in all \*.py files

- added ResObj.getWave()

- added DataMissing exception

- documented that getCurrentModule can return empty strings

- added HOOKERR\_\* to dspeech.h

July 7, 1999

- added getAllUsers

- added ValueError exception type

June 19, 1999

- continued added exceptions for error handling

- added the TestNatLink script for testing this module

- added the logText parameter to displayText

- added getClipboard, used for testing

May 19, 1999

- removed version history from individual files

- changed exception reporting to use exception classes

- added Excepts.h, Excepts.cpp

- ResObj.correction now returns a boolean

May 14, 1999

- added support for allResults

May 4, 1999

- added getWords, correction

- added setTimerCallback

- added getTrainingMode, startTraining, finishTraining

- added createUser, openUser, saveUser, getUserTraining

April 25, 1999

- packaged for external release

March 3, 1999

- initial version

------------------------------------------------------------------------------

This file contains the documentation for the natlink Python extension

module. This extension module provides a thin wrapper around various

functions in the Dragon NaturallySpeaking extended SAPI interface.

playString( keys, flags )

This is a utility function which will send a series of keystrokes to the

window with the focus. The passed parameter is a string in standard

Dragon NaturallySpeaking format (control sequences in braces). This

function will not return until the last keystroke has been drained from

the input queue.

The second parameter is optional. It is a combination of one or more

of the following flags:

0x01 # add SHIFT to the first character in the string

0x02 # add ALT to the first character in the string

0x04 # add CTRL to the first character in the string

0x08 # add RIGHT SHIFT to the first character in the string

0x10 # add RIGHT ALT to the first character in the string

0x20 # add RIGHT CTRL to the first character in the string

0x40 # use the extended keyboard version of the first char

0x100 # set this to defer the termination of the playStrings

operation until the system is sure that the event queue

has been drained. This works around a Win95 problem.

0x200 # set to send system keys, uses low-level keyboard hook

0x400 # when set, an exception will be raises if the user is

holding in a shift key during the playString. This

flag is highly recommended when sending system keys.

0x10000 # use scan codes when generating events

0x20000 # uppercase the entire string

0x40000 # lowercase the entire string

0x80000 # uppercase the first character in the string

displayText( text, isError, logText )

Natlink will create a window in which the user can display messages.

Call this function to append a message to that window, displaying the

window is necessary. The second parameter will cause the text to be

displayed in red instead of black. The third parameter (if 1) will

also copy the text to the dragon.log file.

This function is useful for redirecting stdout and stderr.

Note: this function does nothing unless NatLink is running as a command

and control subsystem for NatSpeak.

getClipboard()

Returns a string which represents the current contents of the clipboard

in text form. The string will be empty if there is no text in the

clipboard.

This function is useful for testing NatLink itself.

getCurrentModule()

This function returns a tuple which contains information about the current

module and window which is active. The tuple contains:

(1) the full file name of the module with the path and extension,

(2) the title of the currently active window,

(3) the handle of the currently active window as an integer.

In the event that the current module name can not be determines, the

getCurrentModule function will return the tuple ("","",0) instead of

raising an exception.

getCurrentUser()

This function will return a tuple of information about the current user

(set of speech files) for Dragon NaturallySpeaking. The tuple contains

(1) the name of the user,

(2) a full directory path where the user's speech files are

located. If no user is loaded, the name of the user will be an

empty string.

getMicState()

This function returns the microphone state, which can be one of 'off',

'on', 'disabled' and 'sleeping'.

setMicState( newState )

Use this function to change the microphone state. You can pass in

'off' to turn the microphone off, 'on' to turn the microphone on (which

takes NatSpeak out of the sleeping state), or 'sleeping' to put NatSpeak

in the sleeping state.

Raised ValueError if the newState is not one of the indicated strings.

execScript( command, args, comment )

This function can be used to execute an arbitrary script using Dragon

NaturallySpeakings built-in script language. The second parameter is

optional and is a list of strings which represents the words which should

be matched to \_arg1, \_arg2, etc. in the script. The third parameter is

also optional and represents a comment which is displayed in an error

messages if the script fails. Usually you pass in the command name.

Raises SyntaxError is there is a syntax error in the script.

getCallbackDepth()

This function was primarily designed as a support function for the

natlinkmain module. This function returns an integer which indicates

how many nested callbacks are active. It returns 0 when you are not

in a callback (for example, during initialization of natlinkmain). For

a basic callback it returns 1 but if you call back into natlink from

a callback causing a nested callback to happen (for example, you call

recognitionMimic) then the callback nesting may be greater than 1.

recognitionMimic( words )

This function simulates the effect of a recognition. You pass in an

array of words which represent the recognition results and NatSpeak

then simulates the exact effect of having recognized that array of words.

You will get an error if you pass in words which are unknown or if the

array of words represents an impossible recognition given the current

system state.

Can raise MimicFailed if the phrase to be recognized is not legal in

the current context (not in an active grammar). You also get this

exception if a word in the input list was invalid.

playEvents( events )

This function is a more powerful version of playString which can play

any sequence of events. You pass in a list of events to play where each

event is a tuple of message, wParam and lParam as defined by windows.

The following tuples are supported:

(0x100, keycode, repeat) # wm\_keydown

(0x101, keycode, repeat) # wm\_keyup

(0x104, keycode, repeat) # wm\_syskeydown

(0x105, keycode, repeat) # wm\_syskeyup

(0x200, x, y) # wm\_mousemove

(0x201, x, y) # wm\_lbuttondown

(0x202, x, y) # wm\_lbuttonup

(0x203, x, y) # wm\_lbuttondblclk

(0x204, x, y) # wm\_rbuttondown

(0x205, x, y) # wm\_rbuttonup

(0x206, x, y) # wm\_rbuttondblclk

(0x207, x, y) # wm\_mbuttondown

(0x208, x, y) # wm\_mbuttonup

(0x209, x, y) # wm\_mbuttondblclk

getCursorPos()

Returns a tuple with the x and y coordinates of the current position of

the mouse on the screen. (0,0) is the upper left corner of the screen.

getScreenSize()

Returns a tuple with the x and y size of the full screen in pixels. This

value can be used to make sure that the cursor remains on the screen.

inputFromFile( fileName, realtime, playlist, uttDetect )

This function will caused Dragon NaturallySpeaking to take its input from

a wave file with the indicated filename. The realtime flag is optional

and if true (integer 1) it will cause the playback to be slowed down to

simulate realtime recognition.

The playlist is optional and represents a list of utterances

(zero-based) to play from the file. The playlist is only valid for

UTT, UTD and NWV files. It is not valid for WAV files. For WAV

files, utterance detection will be used to separate the utterances (all

other files are assumed to have already been separated into

utterances). The play list can contain a mixture of single integers and

ranges represented as tuples of start and ending integers.

Warning: because of a bug in NatSpeak, this code can not handle the

error if you specify an utterance number in the playlist which is not

present in the input file. If that happens you will get a NatSpeak

SDAPI error and this function will not return. The only way to recover

from this error will be to end the Python and NatSpeak processes.

The defatly behavior of inputFromFile is to perform utterance

detection (split speek at pauses) when using files with an extension

of .wav and not perform utterance detection for other file extensions.

You can override this default behavior by setting uttDetect. Setting

uttDetect to 1 forces utterance detection and setting uttDetect to 0

disables utterance detection. The default value is -1.

Raises ValueError is the file is missing or has an unsupported

extension.

setTimerCallback( pCallback, nMilliseconds )

Pass in the address of a Python function which will automatically be

called every N milliseconds. The callback function will not be passed

any parameters. Reset the timer by passing in a function of None.

getTrainingMode()

Returns information about the current training mode. If no special

training mode is active then None is returned. Otherwise, we return

a tuple of the current training mode (one of 'calibrate', 'shorttrain',

'longtrain', and 'batchadapt') and the total number of milliseconds

of speech accepted for training so far.

startTraining( mode )

Initiates a special training mode. The possible training modes are:

'calibrate', 'shorttrain', 'longtrain' and 'batchadapt'.

To perform training (in any training mode), you need to first call

setTraining. Then you need to call the correction method for a series

of previously recognized result objects (ResObj). Finally, after

calling correction for each of the results objects you need to call

finishTraining.

To train a new user from scratch, you need to use the 'calibrate' special

training mode on 5-10 utterances. Then you need to use either the

'shorttrain' or 'longtrain' special training mode. Short training is

appropiate when using BestMatch III models and requires training of

at least 3 minutes of speech. Long training is appropiate for normal

models or the original BestMatch models and requires training of at least

18 minutes of speech.

The third step is to use the 'batchadapt' special training mode.

Usually you retrain the same utterances in 'batchadapt' mode that you

trained in 'shorttrain' or 'longtrain' mode but this is not a requirement.

There is no lower limit for the number of utterances trained using

'batchadapt' mode.

Once a user has been trained it is no longer legal to use the

'calibrate', 'shorttrain' or 'longtrain' special training modes on that

user. However, it is legal to perform additional training using the

'batchadapt' special training mode as many times as desired.

Can raise WrongState you try to set any batch training mode when

calibration has not been done or you try to set calibrate mode if

calibration has already been done. Also if you try to set any

training mode when the system is not in normal training mode.

Can raise ValueError if the mode is not of the listed strings.

finishTraining( bProcess )

Terminates a special training mode. With no parameter, or a parameter

of 1, calling finishTraining will perform the actual training operation.

This may take a while. With a parameter of 0, no training is performed

but the special training mode is cancelled.

Can raise WrongState if a special training mode is not active.

createUser( userName, baseModel, baseTopic )

This call create a new user (NatSpeak speaker profile). The userName

is a required string. The baseModel is an optional string. If

specified it should be the name of one of the build-in base models

(for example: "BestMatch Model"). You can see a list of known models

by running NatSpeak and creating a new user. If missing, the default

base models will be used.

The baseTopic is also an optional string. If specified it should be

the name of one of the built-in base topics (for example: "General

English - BestMatch"). You can see a list of known topics by running

NatSpeak and creating a new user (Professional Edition only). If

missing, the default base topic will be used.

Creating a user does not open it. You need to call openUser separately.

Raises InvalidWord if the user name is invalid.

Raises UserExists if the specified user already exists.

Raises OutOfRange if the specified baseModel or baseTopic does not exist.

openUser( userName )

This call opens a specified NatSpeak user for recognition.

Raises UnknownName is the specified user does not exist.

saveUser()

This call saves any modifications made to the currently open NatSpeak

user to disk.

getUserTraining()

Returns a string which describes whether this user has been trained or

not. Possible values are:

None (which means no training has been done)

'calibrate' (which means only calibration was done)

'trained' (which means the user has been trained)

getAllUsers()

Returns an array of strings of the names of all existing users.

getWordInfo( word, flags )

Looks a word up in the vocabulary and returns an integer which

represents the word formatting information. The word formatting

information can be 0. If the word does not exist in the vocabulary then

None is returned.

Flags is optional (default value is 0). If specified, you can combine

of the following values:

1 = consider inactive words (backup dictionary)

2 = consider active non-dictation words

4 = case insensitive match

The returned integer is a bit vector of the following flags:

0x00000001 - Word was added by the user

0x00000002 - Internal use only

0x00000004 - Internal use only

0x00000008 - Word can not be deleted

0x00000010 - Normally capitalize the next word (like period)

0x00000020 - Always capitalize the next word (like Cap Next)

0x00000040 - Uppercase the next word (like All Caps Next)

0x00000080 - Lowercase the next word (like No Caps Next)

0x00000100 - No space following this word (like left paren)

0x00000200 - Two spaces following this word (like period)

0x00000400 - No spaces between words with this flag set (like with numbers)

0x00000800 - Turn capitalization mode on (like Caps On)

0x00001000 - Turn uppercase mode on (like All Caps On)

0x00002000 - Turn lowercase mode on (like No Caps On)

0x00004000 - Turn off spacing between words (like No Space On)

0x00008000 - Restore normal spacing (like No Space Off)

0x00010000 - Internal use only

0x00020000 - Suppress after a word which ends in a period (like period after elipsis)

0x00040000 - Do not apply formatting to this word (like Cap)

0x00080000 - Do not reset the spacing state (like Cap)

0x00100000 - Do not reset the capitalization state (like close quote)

0x00200000 - No space preceeding this word (like comma)

0x00400000 - Restore normal capitalization (like Caps Off)

0x00800000 - Follow this word with one new line characters (like New-Line)

0x01000000 - Follow this word with two new line characters (like New-Paragraph)

0x02000000 - Do not capitalize this word in a title (like and)

0x04000000 - Internal use only

0x08000000 - Add an extra space following this word (like space-bar)

0x10000000 - Internal use only

0x20000000 - Internal use only

0x40000000 - Word was added by the vocabulary builder.

Raises InvalidWord if the word is invalid.

Raises ValueError if the flags are invalid.

deleteWord( word )

Removes a word from the active vocabulary. The word will still be in

the backup dictionary.

Raises InvalidWord if the word is invalid.

Raises UnknownName if the word is not in the active vocabulary.

addWord( word, wordInfo, pronList )

Adds a word to the active vocabulary. The word may either be

completely new or already in the backup dictionary. The wordInfo is

optional, if missing it defaults to 0x00000001 (word was added by

user).

The third parameter is an optional pronunciation or list of

pronunciations for the word using Dragon System's pronunciation

alphabet. You can add pronunciations to an existing word using

this function but you can not delete pronunciations.

The function returns 1 if the word is added or a pronunciation is

passed in. The function returns 0 if you pass in no pronunciations and

the word already exists.

If the function returns 0 then the wordInfo will be ignored. To change

the wordInfo of an existing word then you must pass in a pronunciation.

For example, use the following trick.

currentProns = natlink.getWordProns( wordName )

natlink.addWord( wordName, newInfo, currentProns[0] )

Usage hints:

- when adding words which are already in the backup dictionary, use a

wordInfo of 0. This mimics Dragon NaturallySpeaking's own behavior.

You can use the getWordInfo function to determine if the word is in the

backup dictionary.

- when adding a single compleyelt new word, include the 0x00000001 bit

in wordInfo. This marks the word as user added and mimics Dragon

NaturallySpeaking's own behavior.

- when adding in lots of words at a time, also set the 0x40000000 bit

in wordInfo. This simulates the effect of using the vocabulary builder

which means that the word will be added with lower frequency.

Raises InvalidWord if the word is invalid.

setWordInfo( word, wordInfo )

Changes the wordInfo for a word already in the active vocabulary.

Raises InvalidWord if the word is invalid.

Raises UnknownName if the word is not in the active vocabulary.

getWordProns( wordName )

Returns a list of pronunciations for the given word. Each pronunciation

is a string. Returns None if the word does not exist.

Raises InvalidWord if the word is invalid.

setTrayIcon( iconName, toolTip, callback )

This function, provided by Jonathan Epstein, will draw an icon in the

tray section of the tackbar.

Pass in the absolute path to a Windows icon file (.ico) or pass in one

of the following predefined names:

'right', 'right2', 'down', 'down2',

'left', 'left2', 'up', 'up2', 'nodir'

You can also pass in an empty string (or nothing) to remove the tray

icon.

The toolTip parameter is optional. It is the text which is displayed

as a tooltip when the mouse is over the tray icon. If missing, a generic

tooltip is used.

The callback parameter is optional. When used, it should be a Python

function which will be called when a mouse event occurs for the tray

icon. The function should take one parameters which is the type of

mouse event:

wm\_lbuttondown, wm\_lbuttonup, wm\_lbuttondblclk, wm\_rbuttondown,

wm\_rbuttonup, wm\_rbuttondblclk, wm\_mbuttondown, wm\_mbuttonup,

or wm\_mbuttondblclk (all defined in natlinkutils)

Raises ValueError if the iconName is invalid.

The following functions are used in the natlinkmain base module. You

should only used these if you are control NatSpeak using the NatLink module

instead of using Python as a command and control subsystem for NatSpeak. In

the later case, users programs should probably not use either of these two

functions because they replace the callback used by the natlinkmain module

which could prevent proper module (re)loading and user changes.

setBeginCallBack( pCallback )

Pass in the address of a Python function to call when the beginning of a

new recognition is detected. Dragon NaturallySpeaking will pause all

recognition processing until the Python function returns. However, it

is safe to call other natlink module functions from within the

Python callback function.

When a recognition starts, this callback will be made and passed a single

parameter which is the same tuple which is returned from getCurrentModule.

setChangeCallback( pCallback )

Pass in the address of a Python function to cal when something in the

system changes. When a change occurs, the callback function will be

passed two parameters. The first parameter is a string indicating what

changed. The second parameter which depends on what changed.

Active user changes: 'user', same tuple returned from getCurrentUser

Mic state changes: 'mic', same string returned from getMicState

The following three functions are designed to be used when you have a

Python program which controls NatSpeak by explicitly importing the natlink

module. Do not use these functions when you are using Python as a command

and control subsystem for NatSpeak. In that case, these functions are

automatically handled for you.

isNatSpeakRunning()

Returns 1 is NatSpeak is running and 0 otherwise. This is the only

function which can be called before calling natConnect.

natConnect( bUseThreads )

Connect with Dragon NaturallySpeaking. This call will launch Dragon

NaturallySpeaking if NaturallySpeaking is not running. As a side

effect of this call, the natlink module will grap a number of COM

interface pointers into Dragon NaturallySpeaking itself. Pass in an

optional boolean to enable thread safety. This is required when running

with the Pythonwin code.

natDisconnect()

Disconnect from NatSpeak by releasing all internal COM interfaces

pointers. This will cause Dragon NaturallySpeaking to stop running it

it was launched by calling connect.

waitForSpeech( timeout )

The Python program must call this function in order to allow speech to

be processed. This function will enter a standard Windows message loop.

When speech events occur, the previously established Python callback

functions will be envoked in a nested call.

A message box will be displayed if the timeout is zero or positive and the

waitForSpeech call will not return until this message box is closed or

until the timeout period (in milliseconds) ends. A timeout of zero

suppresses the timeout. A negative timeout is just like a positive

timeout except that the message box is not displayed (the function only

returns when the timeout elapses).

Inside natlink, each grammar is maintained as a separate object. To expose

this structure, each grammar object will be exposed to Python as a class.

The GrammarBase class defined in natlinkutils encapusulates and extends the

basic functions of a grammar object. It is recommended that you use the

GrammarBase class instead of using GramObj directly. The syntax for

building grammars is documented in gramparser.py.

class GramObj():

load( binary, allResults=0, hypothesis=0 )

Before you use a grammar you need to pass in a binary representation

of the grammar in SAPI format (as a string). This call actually

creates the associated COM objects.

If you set the optional allResults parameter to 1 then you will be

able to get a results object even when the recognition is not

specific to your grammar.

If you optionally set hypothesis to 1 then partial recognition

hypothesises will be available during recognition using a hypothesis

callback (see setHypothesisCallback).

The first DWORD of the SAPI binary grammar format defines the type

of grammar. Command grammars (like those created by GrammarBase

in natlinkutils) have a type of 0. Dictation grammars have a type

of 1. Not every member function of GramObj is supported by every

grammar type.

Can raise InvalidWord if the grammar contains an invalid word.

Can raise BadGrammar if the grammar specification is in error.

unload()

Call this to unload the grammar and delete the associated COM objects.

This function will automatically be called when the GrammarBase object

is garbage collected but it can also be called explicitly.

activate( ruleName, window )

Call this to activate a named exported rule in the grammar. You

can activate multiple exported rules by calling this multiple times.

Pass in the handle of a window (from getCurrentModule) or 0 for

a global grammar.

Pass in an empty ruleName for dictation grammars.

Can raise UnknownName if ruleName is not defined in the grammar.

Can raise BadGrammar if the grammar is too complex to be recognized.

Can raise WrongState if the named rule is already active.

Can raise BadWindow is the specified window does not exist.

deactivate( ruleName )

Call this to deactivate a named exported rule in the grammar.

Can raise WrongState of the named rule is not currently active.

setExclusive( state )

Set the exclusive property on a grammar to force the recognizer

to limit the recognition to only grammars which are marked as

exclusive.

setBeginCallBack( pCallback )

Pass in the address of a Python function to call when the beginning of

a new recognition is detected. Dragon NaturallySpeaking will pause

all recognition processing until the Python function returns.

However, it is safe to call other natlink module functions from

within the Python callback function.

This callback for each grammar object which has been loaded, and after

the global BeginCallback is made.

When a recognition starts, this callback will be made and passed a

single parameter which is the same tuple which is returned from

getCurrentModule

setResultsCallback( pCallback )

Call this to setup a callback function which will be called when a

recognition occurs for this grammar object. Pass the callback

function.

When a recognition occurs, the callback function will be called with

two parameters. The first parameter is the same list which is returned

by ResObj.getResults(0). The second parameter will be a ResObj instance

which represents the results object.

If you have set the optional allResults parameter to load() then

your results callback will also get the results of recognitions

which were not the result of your grammar. If a recognition is

the results of some other grammar in the system then the first

parameter passed to your callback function will be the string 'other'

instead of a list of words. If a recognition corresponds to a

system rejection then the first parameter passed to your callback

function will be 'reject'.

setHypothesisCallback( pCallback )

Call this to setup a callback function which will be called during

the middle of recognitions with the partial hypothesis of the

recognition in progress. You must have set the hypothesis

parameter on load or the callback will not be called.

The callback routine is passed a list of words representing

the best recognition hypothesis so far. The frequency of callbacks

is determined by Dragon NaturallySpeaking. Any word in the

hypothesis is subject to chang ein the next callback or when the

recognition completes.

emptyList( listName )

This function removes all the words in a named list in the grammar.

Can raise UnknownName if listName is not defined in the grammar.

Can raise WrongType is used with dictation grammars.

appendList( listName, word )

This function adds a new word (which can be a phrase) to a named list

defined for the grammar. You can only append one word at a time.

Combine this with emptyList to replace the words in a given list.

Can raise InvalidWord if word list contains an invalid word.

Can raise UnknownName if listName is not defined in the grammar.

Can raise WrongType if used with dictation grammars.

setContext( beforeText, afterText )

For dictation grammars, this sets the speech recognition context.

The context is the set of words (passing in at least 2 words is

suggested) which the recognizer should assume proceed and optionally

follow the text to be recognized next. Pass in two strings (the

strings can be blank) which correspond to text (not words).

AfterText is optional and will be assumed to be blank if missing.

Can raise WrongType if used with command grammars.

setSelectText( text )

For SelectXYZ grammars only, this function is used to tell the

recognition engine what text should be used for the SelectXYZ

grammar. Pass in the visible text exactly as it appears on the

screen. The recognition engine will automatically build the

grammar.

Can raise WrongType if used with other than SelectXYZ grammars.

getSelectText()

For SelectXYZ grammars only, this function returns the text

currently stored in the reocgnition engine for this active

SelectXYZ grammar. getText should return the same value stored

with setText including modifications made by calling changeText.

Can raise WrongType if used with other than SelectXYZ grammars.

Results are also represented by objects which can be manipulated. Result

objects are returned as the second parameter to the grammar results callback.

class ResObj():

getResults( choice )

Call this to return the recognition results for a given choice

on the choice list. Choice 0 (the default) is the actual recognition

results. Other choices are alternatives. Returns None if there

are no results.

Normally returns a a list which which represents the recognition

results. The list will contain a sequence of tuples. Each tuple

will be a pair of the word which was recognized and the rule number

of the innermost rule in the grammar which contains that word.

Note that the rule number is only signifiant for command grammars.

For dictation and selection grammars the rule number is always 0.

Can raise OutOfRange if choice too large for that recognition.

getWords( choice )

Just like getResults except that it only returns a list of words not

rule numbers. Returns None if there are no results for the given

choice. This call is faster that getResults.

Can raise OutOfRange if choice too large for that recognition.

correction( words )

Call this to perform a correction (and adaptation) of a results

object. Pass in an array of words which represents what should

have been recognized (which can be the same as what was recognized).

This call can also be used when doing batch training.

Returns 1 if training succeeds and 0 if training fails. Training

can fail of the transcription is not close enough to the utterance.

This is not considered to be a serious enough error to throw an

exception.

Can raise InvalidWord if word list contains an invalid word.

getWave()

Returns a string which contains binary data which represents the

wave data for this results object.

Can raise DataMissing is no wave data is available.

getWordInfo( choice )

Call this to return the recognition results for a given choice

on the choice list. Choice 0 (the default) is the actual recognition

results. Other choices are alternatives. Returns None if there

are no results.

Returns a list of tuples. Each tuple contains the following elements:

wordName - same as returned by ResObj.getChoice

cfgParse - same as returned by ResObj.getChoice

wordScore - for the first word of the result, this is the score

of this choice; lower is better; zero for other words

startTime - start of word in milliseconds from start of utterance

endTime - end of word in milliseconds from start of utterance

engineInfo - integer (actually a DWORD) properties of the word

wordPron - first pronunciation for the word; not necessarily

the pronunciation which was recognized

Multiply startTime and endTime by 11.025 to index into the wave data

returned by ResObj.getWave

Can raise OutOfRange if choice too large for that recognition.

getSelectInfo( gramObj, choice=0 )

Only for results from the recognition of a SelectXYZ grammar, this

function returns a tuple containing information about the SelectXYZ

recognition.

The first parameter should be the grammar object for which you want

the select info. The section parameter should be the choice number

(just like getResults).

The returned tuple contains:

startPos - character position in the text buffer

(GramObj.getSelectText) for the start of the selection

endPos - character position for the end of the selection

Can raise TypeError is the first parameter is not a GramObj instance.

Can raise BadGrammar if the result did not come from the grammar.

Can raise WrongType if used with other than SelectXYZ results.

Can raise OutOfRange if choice too large for that recognition.

This is a dictation objectm which encapsulates a complete dictation client.

With NatSpeak, when you create a window which supports dictation, you

associate that window with one dictation object.

The dictation object maintains an internal buffer which is an exact copy of

the text in the window. When the user changes the content of the window by

typing, the dictation object buffer needs to be updated. When the user

dictates into the window, the dictation object will change its internal

buffer and then make a callback so the window can be updated.

class DictObj():

activate( window )

Call this to activate the dictation grammar. Pass in the handle of a

window (from getCurrentModule) or 0 for a global grammar.

Can raise BadWindow is the specified window does not exist.

deactivate()

Call this to deactive the dictation grammar. It is safe to call

deactivate even if activate has not been called.

setBeginCallback( pCallback )

Pass in the address of a Python function to call when the beginning of

a new recognition is detected. Dragon NaturallySpeaking will pause

all recognition processing until the Python function returns.

However, it is safe to call other natlink module functions from

within the Python callback function.

This callback is made before the internal buffer is readied for

recognition so it is safe to make changes to the internal buffer

during the callback.

When a recognition starts, this callback will be made and passed a

single parameter which is the same tuple which is returned from

getCurrentModule

setChangeCallback( pCallback )

Pass in the address of a Python function to call when recognition

happens and the contents of the internal buffer is changed in some

way.

The called function will be passed the following parameters:

( delStart, delEnd, text, selStart, selEnd ).

delStart,delEnd - define the range of characters in the buffer which

were replaced by the recognition.

text - is the string which replaced those charcters. The length of

text may be different than end-start.

selStart,selEnd - is the location of the selection after the the

recognition.

The following functions operate on the internal buffer. To be consistant

with Python, it is legal to pass in negative values which are offsets

from the end of the buffer. It is also legal to leave out the end

parameter, in which case the end of the buffer is assumed.

setLock( state )

Call this to control the lock on the internal buffer. setLock(1)

sets the lock and setLock(0) resets the lock. Internally the lock

state is incremented when you set the lock so you need to call

setLock(0) for every (nested) call to setLock(1).

Locking the internal buffer guarentees that the state of the buffer

remains consistant while the lock is held. You should lock the

internal buffer before calling any of the following functions.

Note: NatSpeak requires you to lock the internal buffer before

any of these calls but to make Python programming easier, this

module will set the lock for you. This is ok for a single call,

but if you make multiple calls, you could get inconsistant

information if you are not protected by a lock around all the

calls.

Can raise WrongState if you call setLock(0) when no lock is active.

getLength()

Returns the length of the text in the internal buffer. Note that

getLength maybe somewhat expensive for large buffers.

setText( text, start, end )

Changes the text in the internal buffer at the indicated position.

The length of the region to be changed (end-start) does not have

to match the length of the text. This allows text to be inserted

or deleted.

getText( start, end )

Returns a string which represents the text in the internal buffer

at the indicated position. The length of the returned string will

be end-start.

setTextSel( start, end )

Changes the current selected region.

getTextSel()

Returns the current selected region as a tuple of (start,end).

setVisibleText( start, end )

Establishes a subset of the buffer as being the range of visible

text. Only visible text can be selected or corrected by voice.

Otherwise, the visible text range has no effect on recognition.

Unlike setTextSel which will coerce the end position to match the

end of the buffer, if the start position is 0 and the end position

is missing then the end position will be set to 0x7FFFFFFF rather

than the length of the buffer.

getVisibleText()

Returns the current visible text range which was set by

setVisibleText.

The initial value of the visible text range is 0 - 0x7FFFFFFF

which covers the entire buffer. This ensures that the selection

commands work even if you do not set the visible text range

------------------------------------------------------------------------------

About exceptions in NatLink. NatLink will return exceptions classes for

errors which occur using the NatLink functions. In order to make it

possible to look for specific errors, there are a number of possible error

classes which can be returned. All NatLink error classes are derived from

the base class "NatLink.NatError". The following subclasses are currently

defined.

NatError

Base class for all NatLink errors. This is also returned for

unexpected errors which are not common enough to have been converted

into a more explicit error class.

InvalidWord

Raised when a word or named passed into a function is invalid.

Usually because the word is too long or uses backslash characters

improperly.

UnknownName

Raised when an unknown rule or list names is passed into a function.

Also raised when passing an unknown username to openUser.

OutOfRange

Raised when accessing a result which is not available, usually

because the choice number is too high. Also raised when an unknown

base model or base topic is passed to createUser.

MimicFailed

Raised when recognitionMimic fails for any reason.

BadGrammar

Raised from GramObj.load when the grammar loads for any reason. Also

returned from GramObj.activate when the grammar object is too complex

to be activated.

WrongState

Raised when function can not be applied because system is in the wrong

state. For example, if you call GramObj.activate on a rule which is

already active or you call startTraining('calibrate') when the

speech files have already been calibrated.

BadWindow

Raised when passing an invalid window handle to GramObj.activate.

SyntaxError

Raised when a script with a syntax error is passed to execScript.

UserExists

Raised when calling createUser when the user already exists.

ValueError

Raised when passing an invalid string to setMicState or startTraining.

Also raised when the file passed to inputFromFile is missing or has

an unsupported extension.

DataMissing

Raised by ResObj.getWave when the wave data is not available.

WrongType

Raised by GramObj when you make a function call which is not suppored

for that grammar type. For example, GramObj.setContext is only

supported for dictation grammars; if used on a command grammar this

exception is raised.