COMMUNICATING WITH SPEECH ANALYZER

At start-up Speech Analyzer (SA) is capable of receiving no, a little, or a great deal of information. The state in which SA starts-up is determined by the existence (or lack thereof) and content of the information it receives. When receiving information on which to act, SA may also return information to its caller. It is the intent of this document to define the manner in which and content of, information that can be passed between Speech Analyzer and a calling program.

Speech Analyzer Modes

Speech Analyzer reads its command line to determine in what mode it should operate. If SA is passed nothing or the name of a .WAV file on its command line it operates in a *standalone* mode. That is the mode when SA is run either from Window's File Manager, Explorer or a command prompt. If SA receives the name of a "list" file on its command line then SA operates in a *batch* mode. (The next section further describes list files.) In *batch* mode SA recognizes multiple .WAV files upon which to act.

The Link Between SA and the Outside World

As mentioned above, the manner in which information is communicated to and from SA is through a list file. List files are text files. The content of a list file is the information being communicated. The structure of the list file is the same as that of a standard Windows 3.x initialization file (a .ini file).

Since SA accepts .WAV files and list files on its command line, it needs to know how to differentiate between the two. To make that differentiation SA looks for the command line switch "-L" or "-R" (case in-sensitive) to precede a list file path specification. If SA finds a file path specification with no -L or –R switch preceding it, it assumes the file specification is referring to a .WAV file. Below is the syntax for SA's command-line.

```
SA [[-L|-R] {Screen Size/Coordinates}[ListFileSpecification]]
```

"-L": This specifies SA should start in batch mode and process the *ListFileSpecification*
"-R": This specifies SA should start in batch mode and open in record mode. The *ListFileSpecification* is used to return the names of changed files to the calling application.

Screen Size/Coordinates: This is a window location specification for the main window of SA. It consists of a string of 16 decimal digits. There is no space between the termination of the size string and the *ListFileSpecification.* The interpretation of this field will be discussed further below. As of SA Version 1.5 this field will be optional. Its use is not recommended and will be phased out.

ListFileSpecification: This is a valid DOS file specification and may or may not include the full path. The name of the list file is relatively unimportant to SA, but it is important that when SA returns information to its caller, that it use the same list file name (including path specification if present) to store that information. That will insure that the caller knows what file to look for when attempting to process any changes caused by SA.

Content and Structure of a List File

As noted above, a list file has the same format as a Windows 3.x INI file. This will allow SA and its caller to use the standard .INI processing API calls to read and write information from and to the list file. This also allows for easy future expansion. Following is a description of the sections and entries included in the list file:

[Settings]	This section identifies various settings.
CallingApp=*ApplicationTitle*	The CallingApp entry, so that SA may properly return. The title entry is the string to search for in the applications Titlebar." The title bar will be compared to *ApplicationTitle* + *AnyString.*
WaveOwner=*OwnerName*	The WaveOwner entry will also be used to register files saved as belonging to the calling application. Files which are registered can only be saved if they are opened by the owning Application
ShowWindow=[Hide | Show | Restore | Maximize | Minimize | Size(left, top, width, height) | None]	The "ShowWindow" command provides more flexibility in controlling the size of the SA window. In the absence of this section, SA will size the window using "*Screen Size/Coordinates.*"[1] The options "Hide|Show|Restore|Maximize|Minimize " correspond to the SW_ parameters of the ShowWindow command. The "Size" parameter will size the window using the specified parameters. The "None" parameter will leave the window unchanged.

[1] This is not recommended. *Screen Size/Coordinates* is obsolete.

[DocIDs]
DocID0=*nnnn*
DocID1=*nnnn*
DocID2=*nnnn*
…
…
	This section contains a list of IDs associated with each audio file found in the [AudioFiles] section. This section is provided specifically for Speech Manager. However, any calling program may take advantage of this section. For example, if a database maintains a table of records where each record logs information about an audio file, record IDs could be included in this section so those records could be updated, if necessary, after SA completed its task. Each document ID is a 32 bit number. This list of numbers is pass-through only. SA doesn't need this information except to make sure that it is present when it builds the return list file that its caller may look at. When SA is closed, this section will **only** contain a list of the document IDs associated with the audio files that SA has changed. There must always be one entry in this section for each entry in the [AudioFiles] section.

[AudioFiles]
File0=*AudioFileSpec0*
File1=*AudioFileSpec1*
File2=*AudioFileSpec2*
…
…
	This section contains a list of the audio files which SA should open . Each *AudioFileSpec* must be the full path and file name of an audio file (.WAV) When SA closes, this section will contain a list of **only** those audio files changed (including a full path). There must always be one entry in this section for each entry in the [DocIDs] section. Unchanged filenames will be replaced with the character ":".

[BeginningWAVOffsets]
Offset0=*nnnn*
Offset1=*nnnn*
Offset2=*nnnn*
…
…
	This section is optional and contains a list of the beginning byte offsets into audio file WAV data. Normally, when SA opens an audio file, it displays a window whose left edge represents the beginning of a wave form. If present, these values tell SA to begin viewing (i.e. the left edge of the viewing window) the wave form at the specified offset. There may be one entry for each entry in the [AudioFiles] section.

[EndingWAVOffsets]
Offset0=*nnnn*
Offset1=*nnnn*
Offset2=*nnnn*
…
…
	This section is optional and contains a list of the ending byte offsets into audio file WAV data. Normally, when SA opens it displays, in a window, a default chunk of an audio file's wave form. Providing SA with beginning and ending offsets tells it what chunk of an audio file's wave form to display initially. There must be one entry in this section for each entry in the [BeginningWAVOffsets] section.

[Commands] command0=*CommandName(Params)*	The "commands" section enables background script processing. Each entry represents a command or a command parameter. Each command and parameter includes a sequence number. The sequence number defines the sequence in which commands are executed. The sequence number must start with "0" and increment by "1". All parameters are generally optional. The defaults are underlined. The acceptable values for *CommandName* and associated *Params* will be discussed below.

New Audio Files Returned from SA to its Caller

When SA is opened in *batch* mode it is possible for the user to either make a new recording or open an audio file. When that happens, SA needs some way to tell the calling program that it has processed some audio files that were not specified in the list file. Therefore, the list file that SA returns to the caller contains one new entry under the [DocIDs] and [AudioFiles] section for each new audio file that was opened or recorded in SA. Obviously, the new entry in the [AudioFiles] section contains the full file and path specification of the new audio file. The entry in the [DocIDs] section for the new audio file should specify a Doc ID of zero. Since zero is an invalid Doc ID, the calling program will know that the corresponding audio file is a new one and needs to be submitted to its database for the first time.

Command Specifications

The table below specifies the current *Commands* and *Parameters* as used in the [Commands] section described above. The "#" symbol is used at the end of each command to represent the Sequence number which must be post fixed. Parameters may be omitted provided their ',' separators are present. The default value in absence of certain parameters is shown underlined unless otherwise described.

---

command#=SelectFile(*FileNumber*)

SelectFile selects a file from the [AudioFiles] section by identifier. The *FileNumber* parameter specifies the identifier. If no parameter is included this command has no effect. Commands generally act on the selected file.

---

command#=Import(*szFile,szSegment*)

Import command execute the batch equivalent of the File/Import menu item. The parameters specify which files to import and the mode to use.

szFile correspond to the file name to import. The specification for this file is below. If this parameter is an empty string only segmentation will be performed.

szSegment may take the values [<u>keep</u>|manual|automatic]. These correspond to the values used by Advanced/AutoAlign.

---

command#=SaveFile()

This saves the file without user intervention.

---

command#=Return(bHide)

This command returns to the calling application without user intervention.

The optional bHide parameter is a Boolean which specifies whether to hide SA on return. It may take on the values [0|<u>1</u>].

---

command#=Play(*nSpeed,nVolume,nStart,nStop)*

The "Play" function. This command waits for play to complete before continuing to process commands.

nSpeed [10-50-333]
nVolume [0-50-100]
nStart [0-*FileEnd*] Play start position in bytes
nStop [0-*FileEnd*] Play stop position in bytes

---

command#=DisplayPlot(szType, nTop, nLeft, nWidth, nHeight, nStart, nStop)

DisplayPlot creates a top level plot window of the current file. The plot remains until the window loses focus.

szType [Wave|Pitch | Spectrogram]

nTop,nLeft,nWidth,nHeight together form a rectangle. The rectangle is used for the window and is in screen coordinates.

nStart, nStop specify the regions of the whole graph that is to be displayed.

Command Examples

Examples are included below to make the meaning clearer.

"AutoAlign" command usage

[Commands]
command0=SelectFile(0)
command1=Import(*ExternalFilePath,*keep)
command2=SaveFile()
command3=Return(1)

Play command usage

[Commands]
command0=SelectFile(0)
command1=Play(*Speed,Volume,Start,Stop*)
command2=Return()

ExternalFile Specification

The external file format for import consists of a series of tag delimited entries. Each tag consists of a '\' followed by on or more identifying characters. The following tags are understood:

\ft	Free Translation
\ln	Language Name
\dlct	Dialect
\fam	Family
\id	Ethnologue ID number
\cnt	Country
\reg	Region
\spkr	Speaker Name
\gen	Gender
\nbr	Notebook Reference
\tr	Transcriber
\cmnt	Comments
\table	*Table*

Each tag is optionally included. The order of tags does not matter. Except '\table' must be the last tag. The remainder of the file is included in the *Table.* Unrecognized tags will be ignored. This file should be deleted by Speech Analyzer after it is read.

The *Table* data is composed of a carriage return & line feed, followed by a *Header,* followed by rows of tab delimited data. The *Header* is a tab-delimited set of labels. The following labels are understood.

```
Ref	Etic	Tone	Emic	Ortho	Gloss	POS
```

SPECIFICATION ON HOW SPEECH ANALYZER AND SPEECH MANAGER MANAGE THE PASSING OF CONTROL TO AND FROM EACH OTHER

Using Windows' SendMessage API SA and SM will be able to communicate with each other. The SendMessage API requires four parameters. Below is the text for SendMessage from the Windows 3.1 API help file.

LRESULT **SendMessage**(*hwnd, uMsg, wParam, lParam*)

```
HWND hwnd;          /* handle of destination window   (Integer in VB)  */
UINT uMsg;          /* message to send                (Integer in VB)  */
WPARAM wParam;      /* first message parameter        (Integer in VB)  */
LPARAM lParam;      /* second message parameter       (Long in VB)     */
```

The **SendMessage** function sends the specified message to the given window or windows. The function calls the window procedure for the window and does not return until that window procedure has processed the message. This is in contrast to the PostMessage function, which places (posts) the message in the window's message queue and returns immediately.

Parameter	**Description**
hwnd	Identifies the window to which the message will be sent. If this parameter is HWND_BROADCAST, the message will be sent to all top-level windows, including disabled or invisible unowned windows.
uMsg	Specifies the message to be sent.
wParam	Specifies 16 bits of additional message-dependent information.
lParam	Specifies 32 bits of additional message-dependent information.

Returns

The return value specifies the result of the message processing and depends on the message sent.

Comments

If the message is being sent to another application and the *wParam* or *lParam* parameter is used to pass a handle or pointer to global memory, the memory should be allocated by the GlobalAlloc function using the GMEM_SHARE flag.

Before going further a note should be made about the way in which SM first establishes communication with SA when SA **has not** been loaded. As has been outlined in another document, SM calls SA, by means of a shell command, and passes command-line arguments that tell it to read a list file and what the name of that list file is. What has not be specified before is that when the user chooses to quit SA and return to SM, SA should not terminate itself but it should make itself invisible before relinquishing control to SM. When the user terminates SM and SA is still loaded then a message must be sent from SM to SA telling SA to terminate itself.

When the user attempts the loading of SM when a copy of SA is in memory, SM should display a dialog box telling the user that SM may not be loaded until he terminates that copy of SA. The dialog box will show a continue and cancel button. The continue button is used when the user switches tasks to SA, closes SA and returns to SM where the dialog is still showing. Then he may press the continue button and SM will proceed to finish loading.

When the user attempts to load SA from outside of SM when SM is loaded, then SA should tell the user that that is not allowed and SA should not continue loading.

Neither SM nor SA should allow more than one instance of itself to be loaded at a time.

WHEN SM ACTIVATES SA WHEN SA *IS NOT* ALREADY LOADED IN MEMORY

When SA has not already been loaded in memory then SM needs to call SA and in so doing, pass it some command-line parameters. SA's command-line syntax is as follows:

SA [-l] {*Screen Size/Coordinates*}[*ListFileSpecification*]

SA [-r] {*Screen Size/Coordinates*} [*ListFileSpecification*]

-l	Switch telling SA to open a list file that contains a list of the audio files to process. When this flag is present, there must be a *ListFileSpecification*.
-r	Switch telling SA to open itself in the recorder mode. When this flag is present *ListFileSpecification* is used for returning new file names.
Screen Size/Coordinates	A series of four, four-character numbers representing the size and position of SA's MDI window. When *ListFileSpecification* is included, there should not be any space between it and the 16 characters specifying SA's MDI window size and position.
ListFileSpecification	A full path specification to a file containing a list of files for SA to process. When present the –l or -r flag must be present or the path specification is ignored. When *Screen Size/Coordinates* is included, there should not be any space between it and the list file path and the list file path specification must follow *Screen Size/Coordinates.*

Example: Suppose SM is passing a list file/path specification of **c:\speech\sm\yopno.lst** to SA and SM wants SA to make it's MDI window visible at **100** pixels from the screen's left edge and **35** pixels from the screen top, with a width of **700** pixels and a height of **500** pixels. The command-line to load SA would be:

```
SA  -l   0100003507000500c:\speech\sm\yopno.lst
```

The following would be used to start SA in the recorder mode with an MDI window of the same size and position:

```
SA  -r   0100003507000500
```

(For more information on the *Screen Size/Coordinates* parameter, see the SendMessage example in the section titled "When SM activates SA when SA *is* already loaded in memory *(List File Mode)***.**")

The following sections describe the conditions under which communication via SendMessage is necessary. Also explained are the SendMessage parameters used for the specified communication being described. The same message number of 0x7FF0 should be used for all messages passed back and forth between SM and SA (and later Shoebox). The 16 bit SendMessage parameter should tell the message recipient how to behave when receiving a message of 0x7FF0.

WHEN SM ACTIVATES SA WHEN SA *IS* ALREADY LOADED IN MEMORY *(List File Mode)*

In the event that SA has already been loaded (having been subsequently loaded by SM) and SM needs SA to "wake up" and act upon one or more wave files, SM will send a message to SA telling it to wake up and process a list file. The following parameters should be sent in the SendMessage call.

hWnd	Speech Analyzer's window handle
uMsg	0x7FF0
wparam	0xE001
lparam	Points to a null-terminated string whose first 16 bytes correspond to the MDI window coordinates and size that SA should use when making its MDI window visible. When all 16 bytes contain the character zero (character code 0x30 or '0') then SA should maximize itself. The remaining bytes in the string are the full path and filename of the list file that SA reads for processing. The first 16 bytes are divided up into 4-4 byte chunks as follows.

Bytes 0 - 3	SA's MDI window left coordinate in pixels
Bytes 4 - 7	SA's MDI window top coordinate in pixels
Bytes 8 - 11	SA's MDI window width in pixels
Bytes 12 - 16	SA's MDI window height in pixels

Example: Suppose SM is passing a list file/path specification of **c:\speech\sm\yopno.lst** to SA and SM wants SA to make it's MDI window visible at **100** pixels from the screen's left edge and **35** pixels from the screen top, with a width of **700** pixels and a height of **500** pixels. The *lparam* parameter of SendMessage would point to the following string.

"0100003507000500c:\speech\sm\yopno.lst"

WHEN SM ACTIVATES SA WHEN SA *IS* ALREADY LOADED IN MEMORY *(Recorder Mode)*

When SA has already been loaded and SM needs SA to "wake up" and display its recording functions, SM will should send the following parameters via SendMessage.

hWnd	Speech Analyzer's window handle
uMsg	0x7FF0
wparam	0xE004
lparam	Points to a null-terminated 16 byte (not including the null) string corresponding to the window size and position SA should use when making its MDI window visible. When all 16 bytes contain the character zero (char. code 0x30 or '0') then SA should maximize itself. See the previous section for an explanation of what each of the 16 bytes represents.

When the user is finished using SA and wants to return to SM

In the event that SM has activated SA and after the user has used SA to process one or more documents, SA needs to return control to SM so SM can resubmit to the database any audio files that were changed in SA. When SA returns control to SM, SA should **close all open files,** make itself invisible (not unload itself from memory) and post a message via **PostMessage** that will be received by SM telling SM to "wake-up" and process a list file. Notice that SA should use **PostMessage** and not SendMessage. The parameters for making a **PostMessage** call are identical to those for SendMessage. The following parameters should be sent in the **PostMessage** call:

hWnd	Speech Manager's MDI window handle
uMsg	0x7FF0
wparam	0xE002
lparam	Points to a null-terminated string containing the full file name and path of the list file. If SA made no changes to any of the audio files then *lparam* should return Null. (SA should also, as it currently does, delete the list file from disk when it makes no changes to an audio file.)

Example: Suppose SA is returning the list file/path specification used in the previous example: **c:\speech\sm\yopno.lst**. The *lparam* parameter of PostMessage would point to the following string.

"c:\speech\sm\yopno.lst"

When the user is closing SM and SA is still loaded (but invisible) in memory

In the event that sometime during a user's session of using SM he loads SA to process some audio files, and the user now wants to close SM, a message needs to be sent to SA telling it to terminate itself. If this were not done and SM were closed, there would be no other way for the user to gain access to the loaded copy of SA (except via the task list). The following parameters should be sent in the SendMessage call to tell SA to terminate itself.

hWnd	Speech Analyzer's window handle
uMsg	0x7FF0
wparam	0xE003
lparam	Null

Getting the Window Handle of another Application

When Speech Analyzer and Speech Manager need to get the hWnd of another application then a function similar to the following should be used.

```
////////////////////////////////////////////////////////////////////////////////////////
// This function will return the window handle of the application whose title bar
// contains sAppWinTitle in the first strlen(sAppWinTitle) characters of its title
// text. If no windows contain sAppWinTitle then zero is returned.
////////////////////////////////////////////////////////////////////////////////////////
Int  GetAppshWnd(int iStarthWnd, char *sAppWinTitle)

  HWND ihWnd;
  int  iTitleLen;
  char sTitle[255];

  ///////////////////////////////////////////////////////////////////////////
  // Start off with getting the first top level window in Windows’ window
  // list. Use iStarthWnd (which should be the hWnd of the current app.)
  // to get that window handle.
  ///////////////////////////////////////////////////////////////////////////
  ihWnd = GetWindow(iStarthWnd, GW_HWNDFIRST);

  ///////////////////////////////////////////////////////////////////////////
  // Now cycle through the active windows and get each one's title. If
  // any of the window titles match sAppWinTitle (or sAppWinTitle matches
  // the first strlen(sAppWinTitle) characters) then return true and get out.
  ///////////////////////////////////////////////////////////////////////////
  while (ihWnd) {
  {
    iTitleLen = GetWindowTextLength(ihWnd);

    If GetWindowText(hWnd, sTitle, iTitleLen + 1)
      if !strnicmp(sTitle, sAppWinTitle, strlen(sAppWinTitle))
        return(ihWnd);

    ihWnd = GetWindow(ihWnd, GW_HWNDNEXT);
  }

  return(0);
```