



Matrix² Speech Recognition



I. Introduction

MCCT introduces the addition of Aculab's Connected-Word Speech Recognition (CWR) to the already robust Matrix² product. This value-added feature allows callers to respond to application prompts using either the existing DTMF method or the new Automatic Speech Recognition (ASR) method. This ASR feature broadens selection possibilities and adds a new dimension of interaction between the caller and the IVR. Application programmers will now be able to create Plans that are more comprehensive with less complicated approaches.

This paper presents the ASR feature as it is used in Plan Editor commands with examples of how it can be implemented. Familiarity with the Matrix² Plan Editor and its commands will enhance the reader's understanding of this paper. Please contact MCCT for a copy of the Matrix² User Manual.

II. Aculab ASR Overview

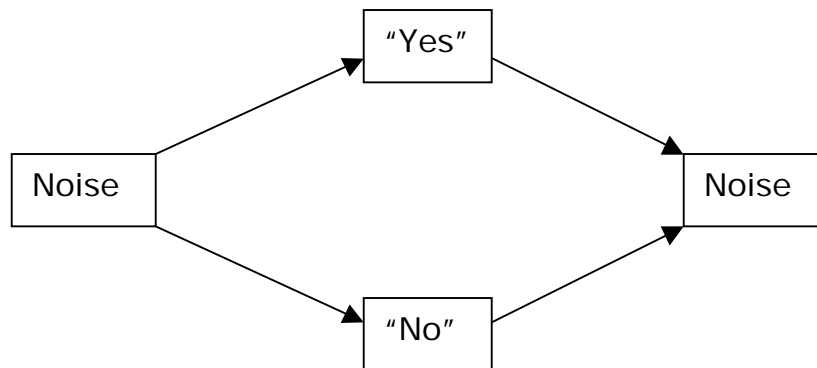
The main elements of Aculab's Automatic Speech Recognition are a phoneme-based recognizer, Connected-Word Speech Recognition server (*ASRserver*) and the 'Grammars'. The terms used will be described in the course of this overview.

A phoneme is a discrete unit of sound in a given language. For example, the word *fish* contains three phonemes written with the Aculab phoneme symbols *f*, *ih* and *Sh* (triphones). An Utterance is a sequence of sounds produced by the caller that consists of phoneme groupings (words) delimited by pauses. Connected-word recognizers process each utterance as a discrete entity. In the case of a caller being prompted to say a 6-digit PIN, each number in the PIN must be separated by a pause that is long enough to be recognized as a delimiter.

The Grammar defining the words to be recognized and their order is specified using Aculab speech grammar format (ASGF) notation – a subset of Java speech grammar format. A discussion of Aculab's speech recognition structure follows.

A *Network*, loaded from a file or built from ASGF, contains a set of nodes and defines the connections between them. Each node refers to a speech recognition model and allows the likelihoods calculated from each model to be accumulated over time. The nodes and their connections are generated from a grammar which is a text string describing the vocabulary which the recognizer must listen for.

As an example, a Network capable of recognizing "yes" and "no" could use a model set containing the two whole word models "yes" and "no", together with an internal noise model to allow for the possible pauses before and after each word.



Exactly two networks are required for a recognition task. One of these defines what may be recognized as spoken data and is termed the "Vocabulary Network". An example of this is the network shown above. The other defines what is recognized as silence and is termed the "Silence Network". A recognition result cannot be generated until the Vocabulary Network outperforms the Silence network. The Silence Network is fixed and must be present in the "net" sub-directory of the ASRserver's working directory.

A Network may be constructed by either using the standalone *ASRNetMan* Windows application (discussed later in this section) or at run time using the appropriate ASR API calls. Each of these methods requires the same inputs (described in the next paragraphs):

- One or more Lexicon and/or Rule objects
- A Model Set object
- An Aculab Speech Grammar Format (ASGF) string

A *Lexicon*, loaded from a file, contains a set of words, each of which is mapped onto one or more alternative sequences of models. These model sequences are specified via sequences of phoneme symbols. There can be more than one pronunciation for each word. Examples are below:

- read: riyd / rehd
- live: lihv / liyv

A Lexicon may be constructed or modified using the standalone ASRLexMan Windows application (discussed later in this section) which can also display a list of the allowed phoneme symbols for each language.

When creating new lexicon entries, it is generally best to use ASRLexMan to look up the pronunciation of a similar-sounding word or words and use a copy-and-paste approach to make up the pronunciation of the new word.

A *Rule*, loaded from a file, contains the rules required to map any word onto a sequence of phonemes (termed “pronunciation by rule”). This method is more comprehensive than any lexicon; it will generate a reasonable pronunciation for any word.

Of course, given the nature of human language it will often make mistakes. That is why rules should only be used as a fallback, for words that are not in the lexicon and are not known at the time the application is designed. It is better to manually extend or customize the lexicon than to rely on the rules since few languages follow an exact set of rules – exceptions are very common.

A *Triphone Map*, loaded from a file (e.g. eng-us-full.rtf) contains a mapping between phonemes and their triphone models, required when building Networks that are to be recognized using an enhanced model file (e.g. eng-us-enhanced.rmf). The triphone mapping will be described further in the discussion on *ASRLexMan*.

Windows User Interfaces

The two major windows applications are *ASRLexMan* and *ASRNetMan*. These may be downloaded from www.aculab.com in the Software Downloads section under Support on the top menu bar (you must register first with a user name and password). Scroll down the page until you come to Automatic Speech Recognition (ASR) and click on that. On the ASR page, click onto the *Windows* folder; download the .zip file and the two .txt files in a new folder created for ASR. Here you can unzip the file that creates nine folders. The executables for each of the two applications are located in folders of the same name (*ASRLexMan* and *ASRNetMan*).

Each application will be discussed for its usage and functionality; double clicking on the .exe files will bring up the blank screens. To use them, you will need to transfer files from the Linux ASR platform to your PC to load into the programs. These files can be found in the /usr/aculab/cwsw/ASRServer directory under three subdirectories: *lex*, *model* and *rul*. This directory structure is shown below:

```

/usr/aculab/cwsw/ASRServer
[root@linux1 ASRServer]# ls
ASRServer          eng-us-full.rtf      model              rul
asr_server.cfg    lex                  net                SessionManager.o
ASRService.o      main.o              nohup.out         Session.o
ConfigFile.o      makefile.ASRServer  orig               svi

```

Files needed for loading the applications are as follows:

- lex:** eng-us-full.rtf (ASRLexMan – Lexicon/ASRNetMan –Pronunciation Files - Load)
- model:** eng-us-basic.rmf (ASRLexMan – Source Files – Models)
- rul:** eng-us-full.rtf (ASRNetMan – Pronunciation Files – Load)

In the ASRNetMan screen, there is a section in the lower left called "ASGF" with a load button. You can create a text file with notepad and rename it as a .asgf file; this is for the purpose of creating a grammar text string for words to use in your application. An example of a grammar text string (or .asgf file) is shown below:

```
(Sussenberger/macomber/dean/john/operator/yes/no/help/one/two/three/four/five/six/seven/eight/nine/zero/star/pound/a/b/c);
```

Placing all of these files discussed above in your main ASR directory makes it convenient to load them using these applications. Each application knows the file types to load, so when a "Load" or "Models" button is clicked, only valid files are shown. The ASRLexMan screen is shown in Figure 1.

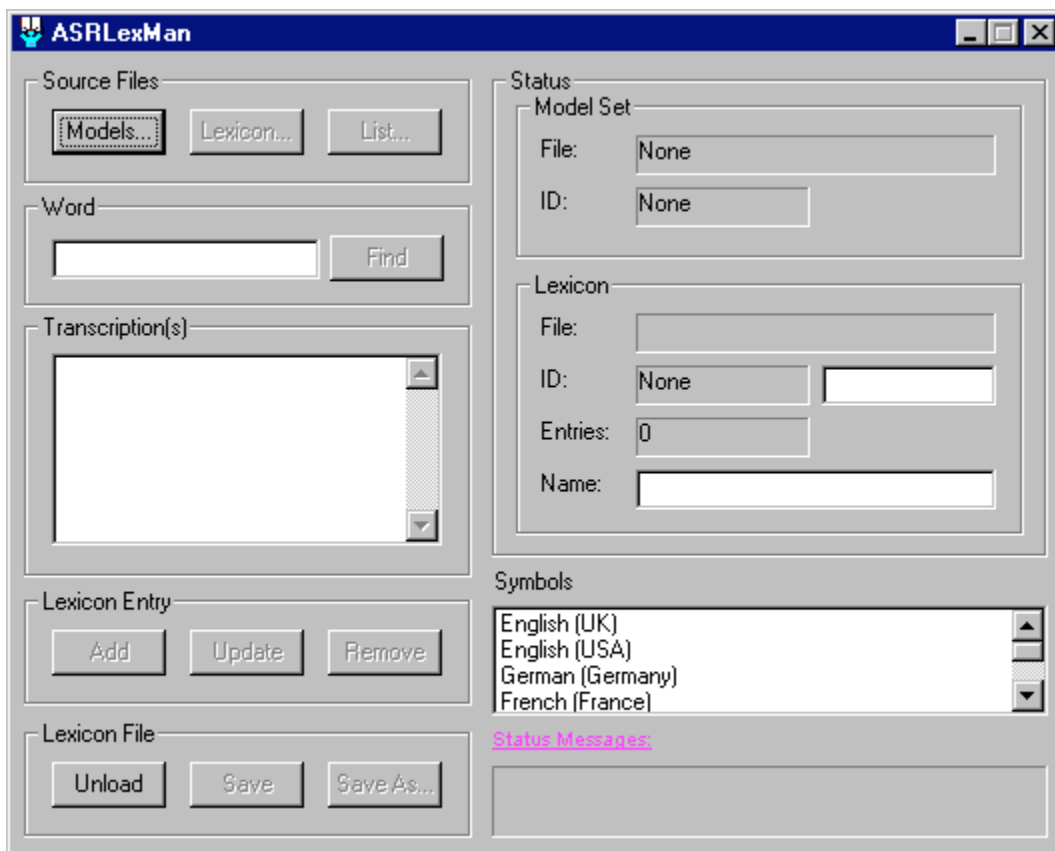


Figure 1. ASRLexMan Screen.

In the ASRLexMan application, the model file must be loaded first; this will make the Lexicon button active. Clicking on the "Models" button shows the valid file to load in Figure 2; all other files that were discussed in the three sub-directories on the previous page are in the same directory.

Note: The file types follow a certain pattern where the middle letter in the three digit extension is key. The .rlf files are *lexicon* files; the .rmf files are *model* files and the .rrf files are *rules* files.

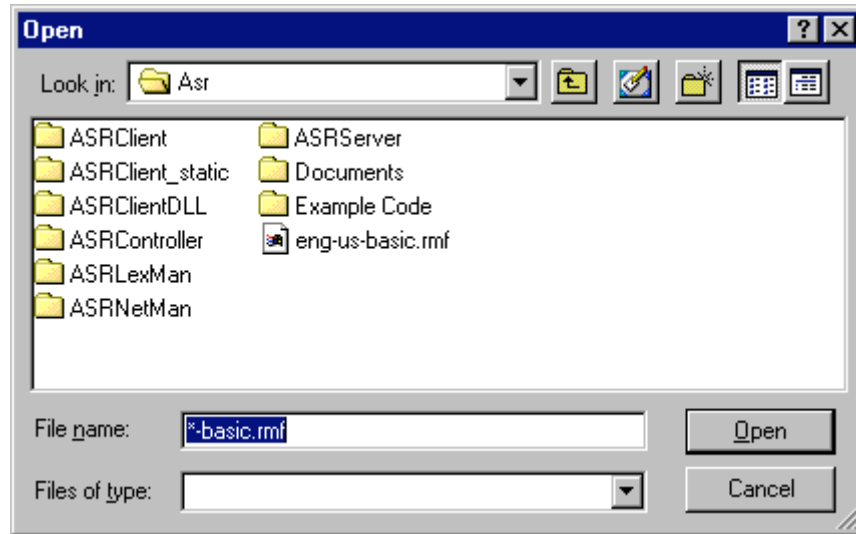


Figure 2. Valid File for “Models” Button.

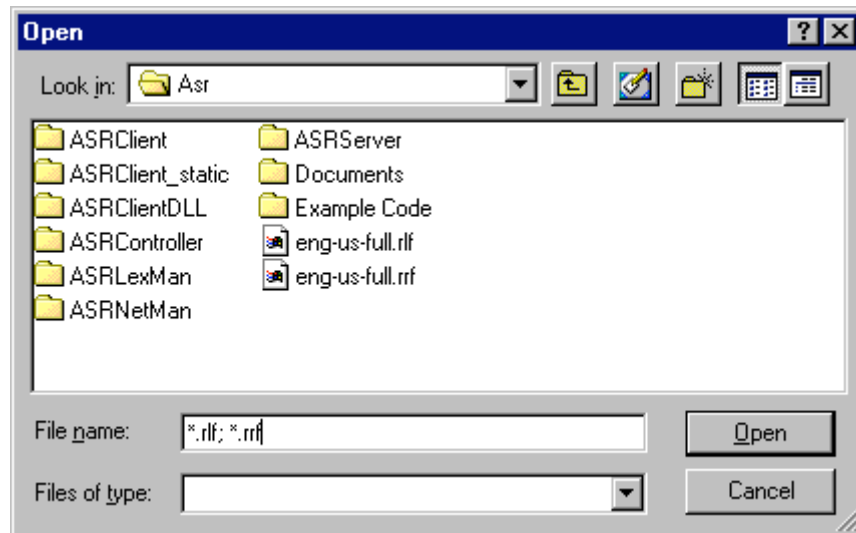


Figure 3. Valid Files for Lexicon Button.

In Figure 3, there are two file types shown: .rlf (Lexicon) and .rfl (Rules). The best approach is to use the .rlf file as this loads the standard ASR phonemes. You may, however, load the .rfl file should you wish to construct your own pronunciation to be recognized. The .rfl file is also used to construct words not contained in the Lexicon.

Figure 4 shows the ASRLexMan screen after the Models and Lexicon buttons are loaded. A search for the word “six” shows the Lexicon pronunciation phoneme that was found. Note: When loading the Lexicon button, it will take a few seconds before the screen loading is complete since there are over 90,000 word entries.

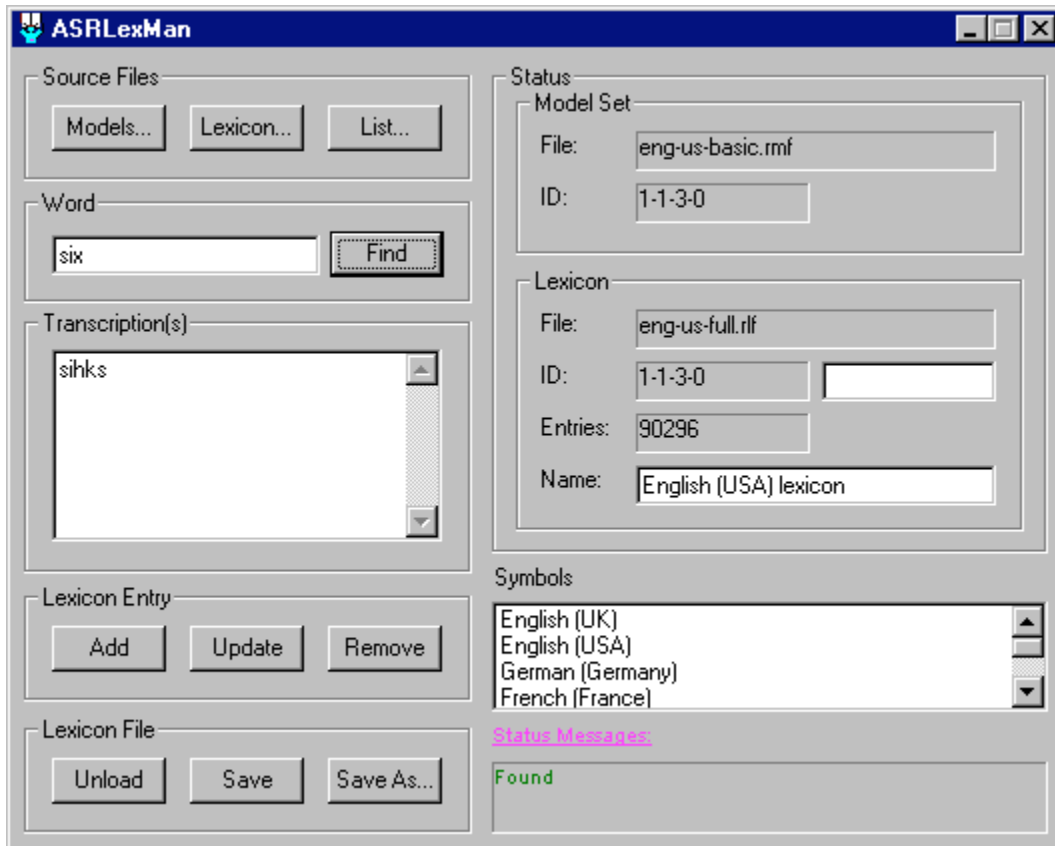


Figure 4. ASRLexMan Screen Loaded.

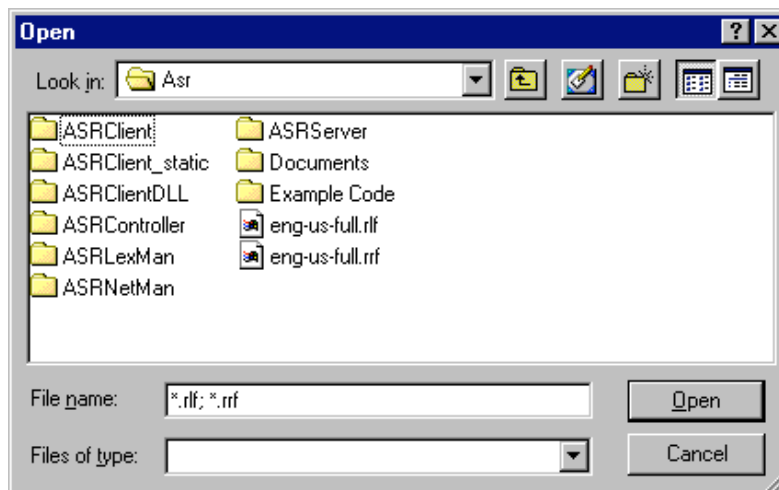


Figure 5. Valid Files for ASRNetMan.

As with the ASRLexMan – Lexicon loading, the valid files for ASRNetMan are .rlf and .rfl as shown in Figure 5. Depending on whether or not you wish to modify or add pronunciation to a word in the Lexicon or use the Rules file to add a word pronunciation to the Lexicon, you will load one of these files at a time.

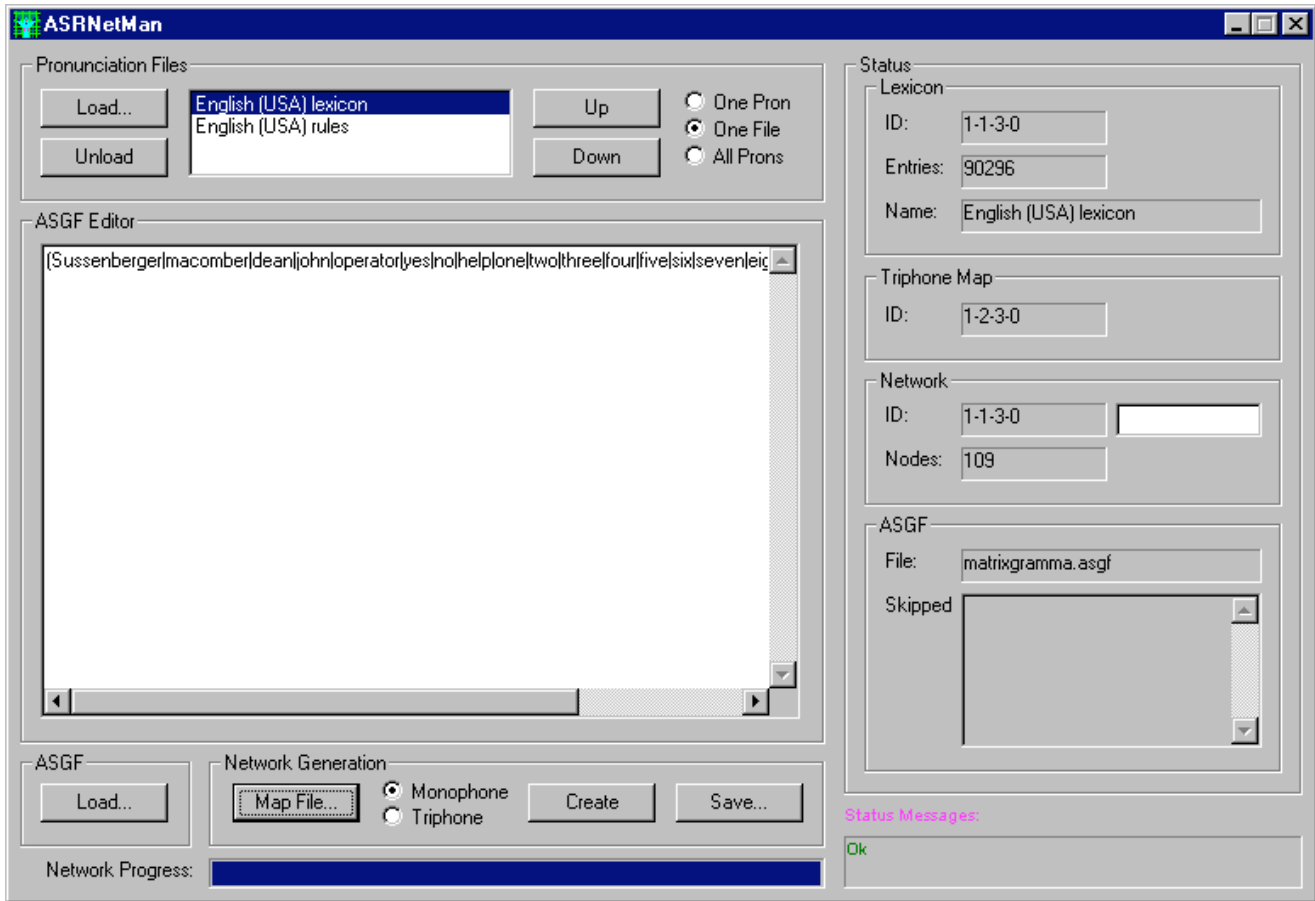


Figure 6. ASRNetMan Screen – Loaded.

Application Usage

ASRLexMan

This application provides the means of managing CWR Lexica. In this context, a lexicon is a list of words, each with a transcription associated with it. A transcription is a list of automatic speech recognition (ASR) models to be used to recognize the word. These models can be a sequence of phonemes (a pronunciation) or one or more whole-word models, or any mixture of the two.

The transcriptions should be entered as a single concatenated string, without any separator. For instance, in US English, the transcription for “hello” is “hehlow”, representing the phoneme sequence “h”, “eh”, “l”, “ow”.

When performing ASR, the incoming audio is processed and then parsed by a recognition network (such as may be produced by ASRNetMan). This Network will have been generated from a grammar definition in ASGF such as “yes | no”, while a lexicon such as *eng-us-full.rlf* will have provided the phonetic transcription of each word in the grammar. In this way, the network contains knowledge of how each word may be pronounced. There can be more than one way of pronouncing a word and all likely possibilities should be included in the Lexicon.

Although Lexica are the preferred method for generating recognition networks, word pronunciation can also be predicted using the Rules provided in files such as *eng-us-full.rrf*. These Rules can be loaded into ASRLexMan in place of the Lexicon. This can give the user an “educated guess” as to the pronunciation of words not explicitly listed in the Lexicon.

After Loading the Models and Lexicon files, ASRLexMan is ready for modifying any lexicon in the file. To look up a transcription for a given word, type that word in the *Word* field and select *Word – Find*. If the word is present in the Lexicon, its transcription(s) will be displayed in the *Transcription* field; if not, the field will remain empty. If a Rule file has been loaded, rather than a Lexicon, the *Transcription* field will display the predicted pronunciation, which the user can then correct (if necessary) and add to a Lexicon by going back to loading the Lexicon file.

To modify the transcription for a given word, or add an alternative pronunciation, first look it up and then modify the contents in the field. While typing, keep an eye on the *Status Messages* to see whether or not your transcription consists of valid symbols. If more than one transcription is required, each should be entered on a new line, i.e., separated by a <CR>. Once the transcription is correct, the entry may be updated by selecting *Lexicon Entry – Update*. Use this same technique in the *Transcription* field if a word lookup fails and you wish to add a pronunciation to a new word in the Lexicon.

To add a list of new words and their transcriptions in an efficient manner, the information may be entered into a list file. Create a text (.txt) file using Notepad and enter all pertinent information. This can be loaded by selecting *Source Files – List*. In this case, when a word is already in the Lexicon, the supplied transcription will be added to the lists of transcriptions associated with that word, unless it is a duplicate in which case it will not be added.

There are many words that are mispronounced in society and provisions can be made in ASRLexMan to allow for this. The word “nuclear” is correctly pronounced as “new klee ar” whereas many say “new kuw lar”. Figures 7 and 8 show how you would add the incorrect pronunciation so that it will be recognized since it is mis-spoken by a multitude of people – including high ranking officials. Figure 7 shows adding the new transcription and the “invalid transcription” message in the *Status Messages* box. By continuing the new transcription to add the last “r”, the message changes back to “valid transcription”. Notice that with an “invalid transcription” message, the *Add* and *Update* buttons are grayed out. By adding the last “r”, the message changes to “valid transcription” and the two buttons become active.

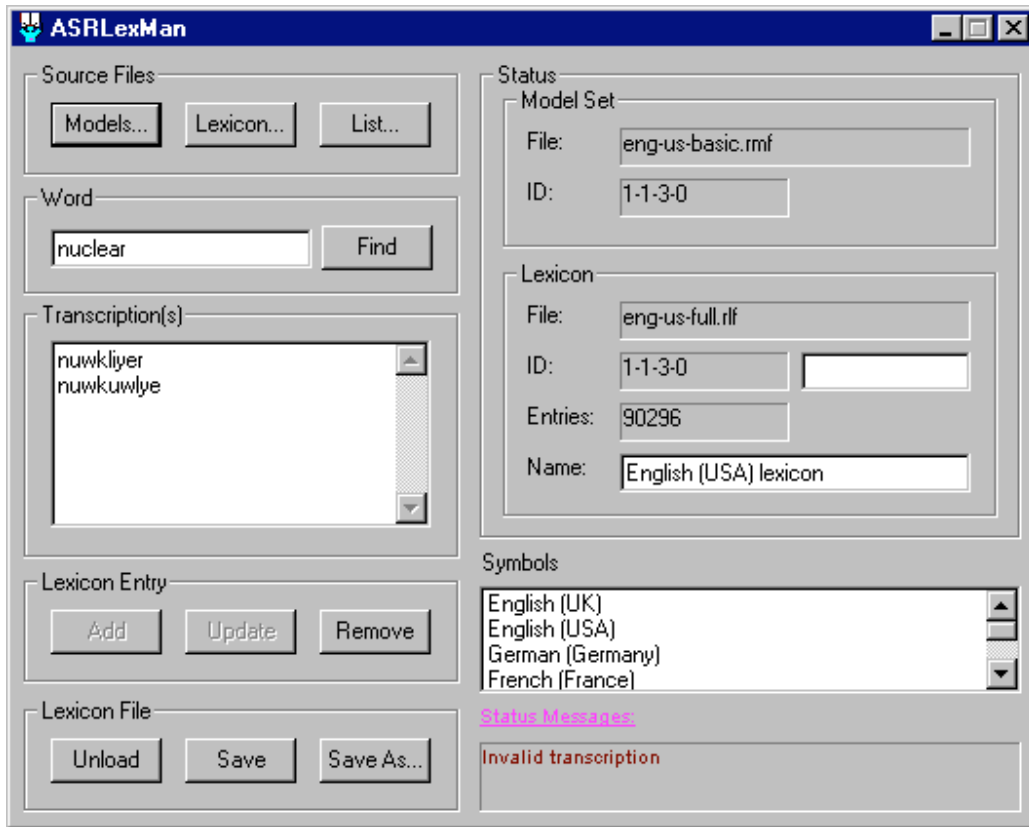


Figure 7. Adding Transcription – Invalid

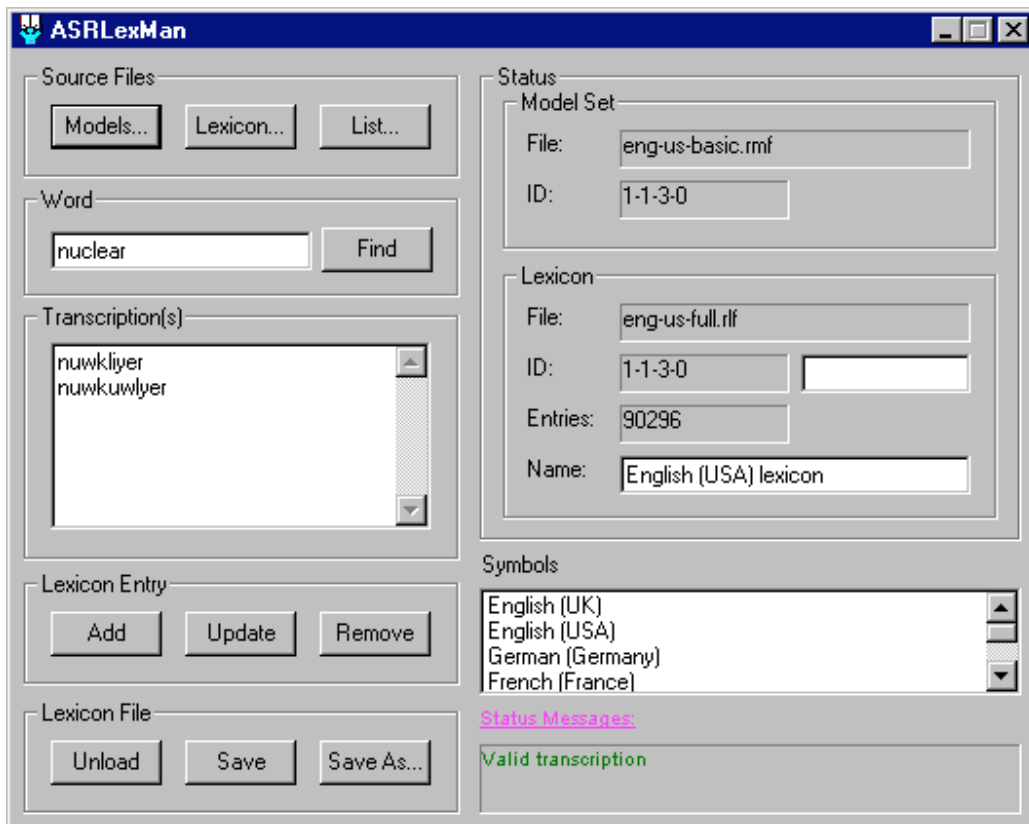


Figure 8. Adding Transcription – Valid

ASRNetMan

The ASRNetMan application allows a user to manage Connected Word Speech Recognition (CWR) networks. A Network is a structure against which processed incoming audio is parsed and recognition results obtained. It is generated from an ASGF (Aculab speech grammar format) definition, one or more lexica, an optional Rule file and (usually) a triphone map file. Figure 6 shows the ASRNetMan screen.

If more than one pronunciation file is loaded, the way in which they are searched may be modified. First, the files are searched in the order in which they appear in the *Pronunciation Files* list; in Figure 6, the files are "English (USA) lexicon" and "English (USA) rules". To move a file up or down in the list, highlight it and click on the "Up" or "Down" button.

If the "All Prons" option is selected, then, for a given word, all transcriptions from all loaded pronunciation files are used as alternative pronunciations. This is only recommended if all the pronunciation files are lexica because the pronunciations predicted by Rule are not always accurate.

If the "One Pron" option is chosen then, for a given word, only the first transcription found *in each file* is used.

If the "One File" option is picked then, for a given word, only the transcription(s) appearing in the first file (in which lookup is successful) is used.

The options "One Pron" and "One File" may be set together so that only a single pronunciation is found.

If a Rule file is included in the list, it is recommended that:

1. It should be last in the list,
2. Only one other pronunciation file should be used, and that should be a Lexicon,
3. "One File" should be set so that the Lexicon is always used in preference to the Rules and
4. One Pron should be cleared so that any multiple entries in the Lexicon are all used, allowing for variation in pronunciation.

As was previously discussed (top of page 4), an ASGF grammar string may be loaded. This can be entered manually in the *ASGF Editor* field or a text file with an extension of .asgf may be loaded using *ASGF – Load*.

Note: You can create a .txt file in Notepad, save it and rename it from filename.txt to filename.asgf.

Building the Network is the next step. If triphone models are to be used at run-time, select *Network Generation – Map File* to load a triphone map file. For US English this should be *eng-us-enhanced.rtf*. Normally, the *Network Generation – Triphone* option should then be selected; otherwise the Network will be built to work with *monophone* models.

If no triphone map file is loaded, a monophone network can still be built, but that should only be considered if there are severe system constraints (triphone models give much higher recognition accuracy).

The Network is built by selecting *Network Generation – Create*. If the Network is large or complicated, look at the *Network Progress* bar to monitor progress and *Status Messages* to see what is happening and if any warnings or errors were generated. If any words are not found in the lookup, they are skipped and, as far as possible, the Network will have been built around them.

Any skipped words will be listed in the *ASGF Skipped* field. However, if any words are listed there they must be treated as an error and either:

- The grammar should be edited to correct any spelling mistake(s),
- The Lexicon should be extended with ASRLexMan to include the skipped word(s) or
- A Rule file should be added to the end of the pronunciation file list to predict the pronunciation of any out-of-lexicon word(s).

Once the Network creation is complete, you must save it using *Network Generation – Save*. Figure 9 shows a saved triphone Network.

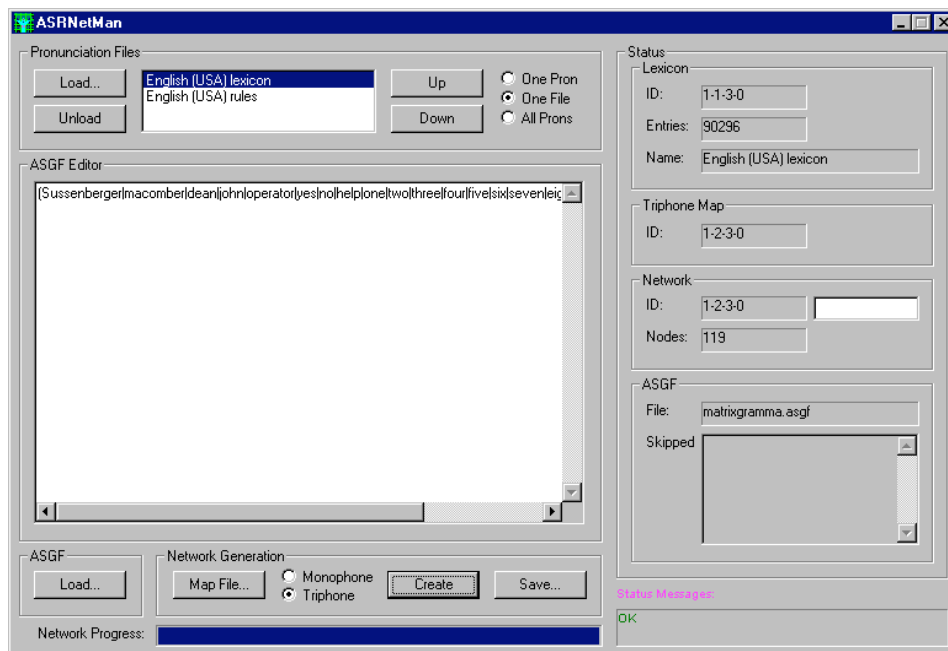


Figure 9. Saved Triphone Network

III. Matrix² Speech Recognition Implementation

The Matrix² Plan Editor contains a multitude of commands for creating a “Plan” or an application. Initially, some of these commands were designed to collect DTMF from the caller for such things as selecting an option, entering a PIN or credit card number and so on. The two most common Matrix² commands for collecting DTMF are: *Select* and *GetDTMF*. These command screens are shown in Figures 10 and 11, respectively.

Program Flow SELECT Command

Display Label

Parameters

Prompt Names

To Be Played

Store DTMF In Prompt On Invalid Key

No. Of Retries Interruptible Message? No

On Maximum Retries Go To Plan To Label

Key	Go To Plan	Go To Label	Key	Go To Plan	Go To Label	Key	Go To Plan	Go To Label
0	<input type="text"/>	<input type="text"/>	4	<input type="text"/>	<input type="text"/>	8	<input type="text"/>	<input type="text"/>
1	<input type="text"/>	<input type="text"/>	5	<input type="text"/>	<input type="text"/>	9	<input type="text"/>	<input type="text"/>
2	<input type="text"/>	<input type="text"/>	6	<input type="text"/>	<input type="text"/>	*	<input type="text"/>	<input type="text"/>
3	<input type="text"/>	<input type="text"/>	7	<input type="text"/>	<input type="text"/>	#	<input type="text"/>	<input type="text"/>

Voice Recognition

No. Of Timeouts Waiting Prompt During Timeout

Timeout Interval Timeout Go To Plan To Label

Disconnect Voice Resource? Yes

Previous Save Cancel Next

Figure 10. *Select* Command Screen.

Figure 11. *GetDTMF* Command Screen.

The *GetDTMF* command is the same as the original; no fields have been added to allow for Speech Recognition values at this time. This is currently handled by another means and will be discussed later in this paper. The *Select* command has had the addition of the line shown between the red arrows in Figure 10. This new line can contain a string of words separated by a “|” delimiter; these words are what make up the valid “Grammar” words for this Plan or application.

***Select* Command**

In the DTMF-only mode, the *Select* command was limited to twelve choices per command: 0 – 9, * and #. More choices required additional *Select* commands. Using an example of an application where callers can select one of thirty ring tones to be downloaded based on their account information; in the DTMF-only mode three *Select* commands would be required. With Speech Recognition, this can be done in one *Select* command with a single voice prompt. In multiple *Select* commands, three separate voice prompts would be required.

In the next few pages, an example of how the *Select* command can be used with Speech Recognition will be shown. This will involve other Matrix² Plan Editor commands to show program flow and functionality. For more details on the Plan Editor commands, please contact MCCT Inc. for a Matrix² User Manual.

The example involves callers in a Plan where they can purchase items and have them shipped to their home. Each caller will have an account with billing and ship to information based on their PIN and user name. The items for this example are team pennants for major league baseball teams. In the *Select* command, team names with two words will be chosen in the 0 – 9, *, # section. All other team names that are single words will be chosen by saying the team name; this will be set up in the new 'Voice Recognition' line.

Program Flow SELECT Command

Display Label

Parameters

Prompt Names

To Be Played

Store DTMF In Prompt On Invalid Key

No.Of Retries Interruptible Message?

On Maximum Retries Go To Plan To Label

Key	Go To Plan	Go To Label	Key	Go To Plan	Go To Label	Key	Go To Plan	Go To Label
0	<input type="text"/>	<input type="text"/>	4	<input type="text" value="White_Sox"/>	<input type="text"/>	8	<input type="text"/>	<input type="text"/>
1	<input type="text"/>	<input type="text" value="Red_Sox"/>	5	<input type="text"/>	<input type="text"/>	9	<input type="text"/>	<input type="text"/>
2	<input type="text"/>	<input type="text" value="Blue_Jays"/>	6	<input type="text"/>	<input type="text"/>	*	<input type="text"/>	<input type="text"/>
3	<input type="text"/>	<input type="text" value="Devil_Rays"/>	7	<input type="text"/>	<input type="text"/>	#	<input type="text"/>	<input type="text"/>

Voice Recognition

No.Of Timeouts Waiting Prompt During Timeout

Timeout Interval Timeout Go To Plan To Label

Disconnect Voice Resource?

Figure 12. Example Select Command.

The voice prompt *Pennant* is shown in Figure 13 on the following page.

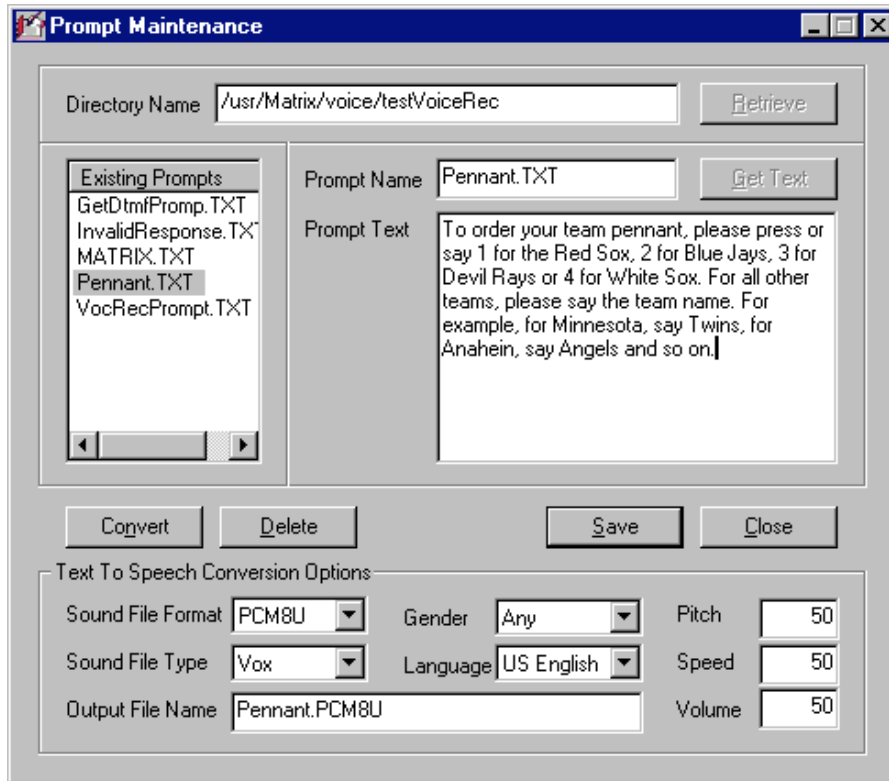


Figure 13. Voice Prompt for *Pennant?* Select Command.

For teams with single names, the program flow goes to a series of *IF* commands to add the chosen team name to the callers account to bill and ship. Figure 14 shows the first *IF* command.

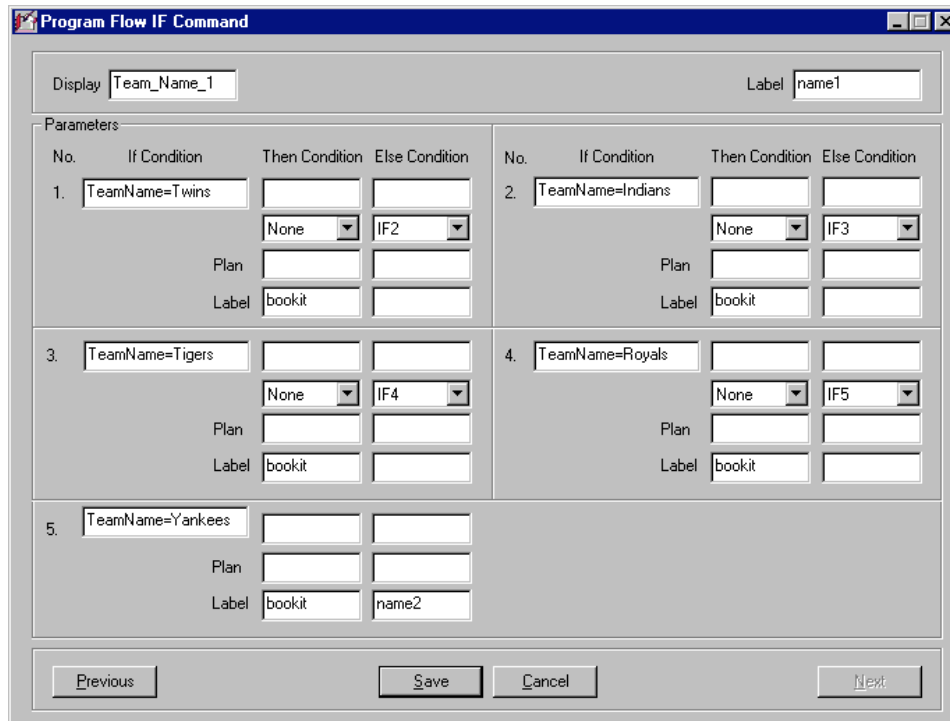


Figure 14. First in a Series of *IF* Commands.

In the *Select* command shown in Figure 12, the name spoken for the team name is put in the variable *TeamName* (the values for 0 – 9, * and # are replaced using another command as will be explained later). In the series of *IF* commands, the value for this variable is tested and when the actual name is determined, it is carried to a database command where the variable value is added to the callers account. (Since the purpose of this paper is to discuss Matrix² Speech Recognition, the entire flow of this Plan will not be discussed as that is a Plan Editor topic).

To put the proper values in the *TeamName* variable for the 0 – 9, * and # choices, each of these is sent to a Label with the two word team name and assigned the correct value. This is done using a *SetVar* command as shown in Figure 15.

Variable Names	Variables Values	Variable Names	Variables Values
TeamName	Red Sox		

Figure 15. Setting Correct TeamName with SetVar

Each *SetVar* command is followed by a *GoTo* command to take the flow to the first *IF* command – *name1*.

GetDTMF

The *GetDTMF* command behaves in a similar manner to the 'Voice Recognition' portion of the *Select* command. As in the *Select* command, any retrieved word, number or series of numbers is stored in the 'Store DTMF In' variable. The value that is stored in this variable can then be retrieved and acted upon by another command.

In the *Select* command you can store valid words in the 'Voice Recognition' field. At the time of this writing, the only anticipated values are numbers – either single digit or number strings such as PIN's or credit card numbers. It is possible that future development efforts will add this 'Voice Recognition' field to the *GetDTMF* command if it is deemed necessary. For the time being, however, any word that is contained in the Grammar Lexicon (refer to the *ASRLexMan* section of this paper) is valid for the *GetDTMF* command. Foreseeable uses are the spelling out of an e-mail address or gathering street address, city, state and country information. Using the *ASRLexMan* utility will allow these options to be achieved.

IV. Summary

The new Matrix² Speech Recognition feature adds a tremendous advantage to the already powerful Matrix Plan Editor. Some examples have been shown here, but the only limits are one's imagination. It has been shown that ASR adds much more power and flexibility to the *Select* command providing more options to the caller in a less cumbersome way. More information can now be collected in a single call that goes way beyond the limitations of a touch-tone keypad.

If you have any questions or comments regarding this paper, please contact MCCT at 800 GO-4-MCCT (US), 603 524-2214 (International) or info@mcct.com.