Project Number: CS-MLC-VR98

VAMP – A Voice Activated Music Processor

A Major Qualifying Project Report
submitted to the Faculty
of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the
Degree of Bachelor of Science

by

_____
Kristopher T. Babic

_____
Daniel P. Hebda

_____
Brian A. Whitman

Date: December 18, 1988

Approved:

_____
Professor Mark Claypool, Major Advisor

1. voice recognition
2. computer music
3. user interface

Abstract

This project developed and described a tool to control complex musical systems with voice recognition. VAMP – A Voice Activated Music Processor, implements voice control as a user interface, a music object representation system, and a musical meaning parser tied into a prototyped computer sequencer. With VAMP as a framework, composers and musicians can gain control of the complex dynamic properties of computer-controlled music by using their voice.

Acknowledgements & Notes

The VAMP group would like to thank their advisors, Professor Mark Claypool and Professor Frederick Bianchi, for their guidance, help, and support.

They would also like to thank Tim Thompson, the developer of KeyKit, for his help during the final stages of the project.

All work on this Major Qualifying Project was done equally by all members of the group throughout the course of the project. Individually, Kris Babic concentrated on the Voice Recognition module, Daniel Hebda the Parser module, and Brian Whitman the Output / KeyKit module.

Table of Contents

Table of Figures

1. Introduction

Since humans have communicated for thousands of years using language, we find it the easiest to discuss, recommend, and control by merely talking. When computers and other information systems appeared to the public in the mid-twentieth century, interfaces were designed not with the user in mind but the machine: it was easier (and more possible) for an engineer to install four red switches and two blinking lights than to have a text-based entry system. And as computers became more powerful, that paradigm remained: the switches evolved to punch cards which evolved to the keyboard, but even then the user had to work at the machine's level. The earliest home computers had their operators loading their spreadsheets or word processor using arcane disk commands, and it was not until the advent of the sixteen-bit operating systems publicized by Apple (the Macintosh), Atari (the ST series) and Commodore (the Amiga) that we were then freed from the keyboard to use a much more natural mouse and window system.

Since then we have been only slightly improving on this model: the latest interfaces are merely redesigns of a redesign. The public's eventual goal, as so prophesized by that ubiquitous barometer of technical advancements, HAL from the film 2001, is the natural voice recognition interface. We would like to finally regain control of our machines and have them try to understand *us* instead of the other way around.

Music has undoubtedly benefited a great deal from the introduction of computers: from the first IBM machines and into the Internet, music composition and creation has been an artist's "killer

application." Music is notable for being both a history-long means of expression and also mathematically describable. A composer that had to previously hand copy a score for each part now loads up a scoring program. Sequencers control virtual instruments in real time, allowing for quantization and step recording. Even the electronic age has affected the art: a musician can buy programs that algorithmically generate music for as-yet-unheard patterns of tones and sounds, and digital processing, emulation and recording software has expanded the artist's palette to never-before attainable variety. No competitive recording studio is without a computer system complementing its 'traditional' instruments.

However, with all of these options available to a user, it becomes daunting to try to control them all. Many studios have dozens of sound modules with hundreds of sounds in each, all with hundreds of parameters that are called real-time from the computer system. For a user to try to control them all is an exercise in futility, and most sequencers do not let users modify the user interface to their tastes.

Since Voice Recognition has recently become possible, and the challenge and deadline HAL presented us might just seem viable, a computer-based music sequencer controlled by voice emerges as a natural solution to the problems listed above. With this solution, a composer can control multiple dynamics at once as simply as they would talk to an orchestra. Variations that would be impossible to do in real time with a mouse, such as lower the volume on all tracks with a violin on the third measure, would be feasible without causing a break in the music. This is important for power users, since by adding a voice system control, a user can convey a much larger amount of information than by just using the mouse and keyboard combination. Since MIDI studios

have only gotten more complex, with multiple samplers and sound modules linked to the computer, the prospect of voice control enables a higher level of efficiency from the user that has difficulty controlling these all at once.

VAMP, a Voice Activated Music Processor, was developed out of these issues. The VAMP project team developed a framework for controlling music commands through speech control. This following paper discusses VAMP, a product that parses the speech of a conductor and outputs it to a musical object language that a sequencer then understands.

The contributions of this project are:

1. The development of a parser for a musical grammar. Musical commands that a conductor or composer would make are different from 'standard' English, since the object and attributes have different properties. The Voice Activated Music Processor (VAMP) includes a parser that identifies musical objects in a series of groups that are then sent to the music module.

2. The creation and implementation of a musical object representation. VAMP quickly and simply integrates the parsed voice with the musical module by using intermediate musical objects. This representation is generic and abstract enough to be upgraded and implemented on a number of different platforms.

3. The application of voice recognition to a user interface. While many advancements have been made in this field, it is still relatively new due to recent dramatic increases in computing speed

and capacity. Our voice recognition (VR) interface allows a user to control a large set of musical objects by naturally voicing the command.

4. The development of a MIDI sequencer to demonstrate these goals. VAMP makes use of the development kit "KeyKit" that allows VAMP and the parser to get at the music data within a sequence. This sequencer link, while developed solely for demonstration purposes, is a powerful realization of the tenets of this project.

2. Related Work

Voice control systems and interactive music devices are not a new field, but rather, the synthesis of them is what makes this project unique. However, to fully understand the context of VAMP, one should familiarize oneself with the background relating to these two topics.

2.1 Voice Control Applications

The promotional material that Dragon Systems, Inc. lists on their packages champions many benefits towards voice recognition software. Voice Recognition "frees users hands and eyes for other tasks… while simultaneously inputting data," "simplifies computing for novice users," "improves data entry speed and accuracy," "automates processes requiring instant data access," and "protects workers against repetitive stress injuries." Those benefits are what is driving a new industry that hopes to replace the keyboard or mouse as a standard means of inputting data and controlling a computer.

Voice control applications come in various forms. The end-user packages, which retail from $50 to $250 (USD), contain an learning program and hooks for a word processor. The learning program runs the user through positioning a microphone and setting up the computer's audio subsystem. Usually, the user will spend a nominal amount of time (an hour or two) reading text from the screen while the software analyzes the voice input. Once that initial training is completed, the user can start dictation, but usually with poor results. Dictation occurs by starting a word processor that has hooks for the particular voice control package and merely speaking into the microphone.

However, the more time the user spends talking to the computer, the better it will understand the voice in the end. Even while dictation is occurring (not training), the voice subsystem is updating its 'voice print,' or set of files that contain user voice information. So at first, VR systems seem to provide users only an endless amount of errors and problems, but as a patient user continues to use the system, rewards soon come in the form of up to (Dragon Systems package claimed) 95% accuracy.

The January 1998 issue of Byte magazine features a roundup of the currently commercially available voice recognition packages. At the time of the writing, Dragon's NaturallySpeaking and IBM's ViaVoice were the two high-profile players. In real world situations, the reviewer rated Dragon as having a 10.4% error rate after training, while Via Voice had a 13.3% rate. (Kay) Other packages exist at a lower cost (and geared to simpler dictation needs). This particular review claims Dragon's accuracy a step above ViaVoice's, but both are essentially similar programs.

Dragon Systems and other companies, perhaps sensing the future market for voice-controlled user applications, supplies their customers with a development package based on the voice dictation engine. A programmer inserts a custom control (essentially a C++ class) into their code and can then access the speech output of the voice recognition system. Microsoft, Inc. has standardized this output and called it SAPI (Speech Applications Programming Interface). This way, the user can choose in the end which engine to use with their voice-controlled applications.

Other voice systems do exist, but they are on the industry scale, many costing tens of thousands of dollars for commerce applications and operator routing. They also run on proprietary hardware. Voice control, being that is a natural extension of human communication, has been the holy grail of user interface designers, but only recently has it arrived practically for the home computer user. This project is one of many that has recently begun to practically apply voice control to enhance a user interface.

2.2 Sequencers / Interactive Music Systems

Computer music sequencers have been around since the advent of MIDI, the standard that allows musical instruments and computers to interchange music data on a serial bus. Ikutaro Kakehashi, Roland's president, saw the need for a standard among all instruments in 1981. He communicated with Tom Oberheim and Dave Smith from Sequential Circuits, and by 1983, the MIDI 1.0 specifications were released. (Chadabe, 195) MIDI paved the way for a large number of instruments and computer programs that control music, due to the cross-platform standard. It didn't take long for computer programmers to realize that MIDI could be perfect for storing note data and controlling synthesizers and other sound modules by a piece of software. Users could be able to edit notes on screen and effortlessly modify and compose music.

Out of the short history of computer music sequencers that were to follow the advent of MIDI, our present state includes two or three "major players" on each platform (PC and Macintosh). The PC platform offers Twelve Tone System's "Cakewalk," geared towards home studio users and

hobbyists, or Steinberg's "Cubase." The Macintosh platform consists of "Digital Performer" or "Studio Vision Pro," as well as Cubase. Sequencers have evolved from merely acting as a database for recorded MIDI data to being full-fledged workstations. Most also now allow a user to sequence digital audio as well as MIDI data.

While the glut of sequencers available to the musician offers a large degree of choice, it also confuses: most of the software above contains such wildly different user interfaces that it would take weeks to migrate from one package to another. This is because music doesn't transfer well over to a windows-icons interface; it is classically a 'hands on' art. Realizing this, many music manufacturers produce "MIDI Controllers," hardware boxes that allow MIDI signals to be sent from the user in a much different fashion than the keyboard-mouse way. For example, Peavey Electronics, Inc. produces a controller with 16 hardware sliders: when "mixing down" a virtual orchestra, a user can move the sliders instead of clicking on virtual sliders and dragging them on screen. Other devices mimic instruments: one can purchase breath controllers for lifelike horn dynamics, or simulated drum kits with MIDI-triggered pads instead of drum heads.

Robert Rowe's Interactive Music Systems: Machine Listening and Composing outlines a new science for musicians and computer scientists: machine control of sound and music. Throughout the book, Rowe describes systems created on a development package for Macintosh, Opcode's "Max" that control various music and sound parameters through graphically described algorithms. The systems he creates and describe mostly deal with computer-generated music, such as fractal algorithms and other math-oriented composition, but the general idea is that of "no barriers:" with

Rowe's ideas, coupled with the current state of the technology, it is easy to see the wealth of opportunities these systems can offer us.

Software such as Max and its Windows counterpart Pure Data (Pd) allow a user to design a controlled system by linking together various modules. Each module is treated as an object with its own methods, parameters and output. Many proprietary "studio-specific" music problems a composer might be having can easily be solved by using Max to model the system. Other similar software, such as KeyKit (which is described in greater detail later) allows a user to actually program the tools they need in a C-like programming language. This allows for greater control, but is not as easy to understand and implement.

Some systems use a musical language representation to allow a developer to get at the data within. These formalized grammars appear close to C++ in that they are object oriented with traces of inheritance. A user controlling a package with such a grammar can understand the system easier, since musical data is abstracted to a hierarchy much like music itself; notes are values with properties that can be placed in groups of phrases or measures, which can be arranged into song objects with their own properties as well.

The work done on these systems up to this point have been geared to the scientific community: Max has yet to be accepted as a common tool for popular music composition. But as the need for more complex systems grow, such as VAMP, interactive control of music by development-like music "environments" will take center stage.

3. Approach to Project

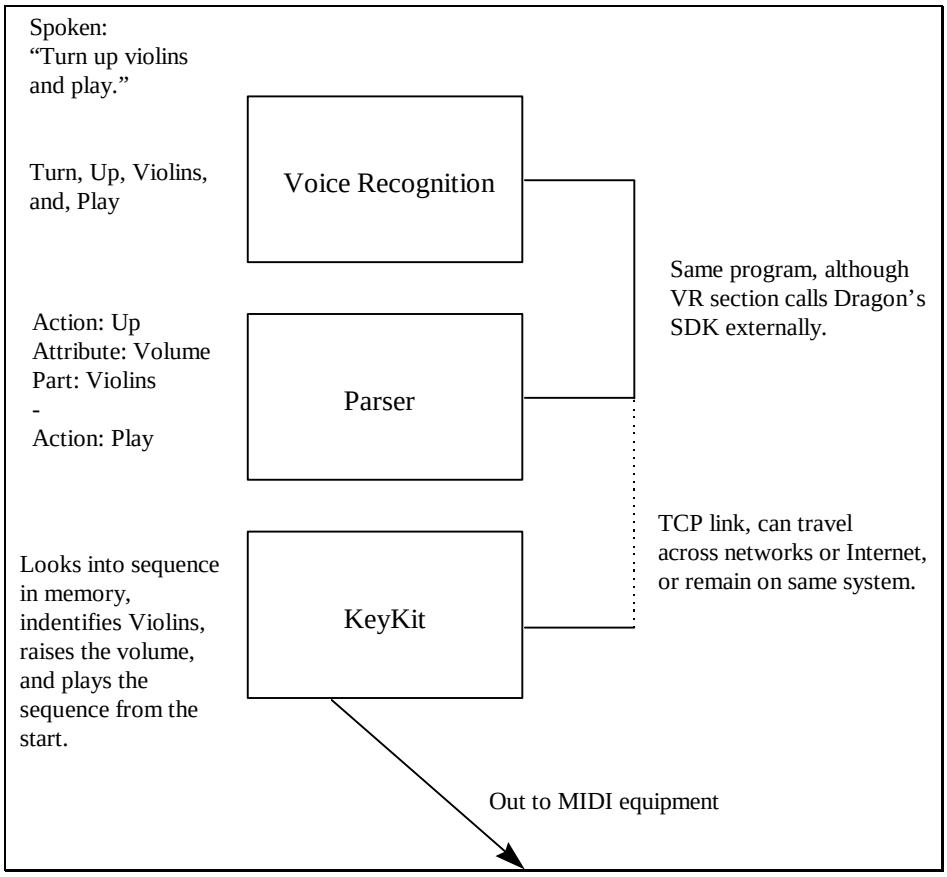

Figure 1 - The Three Module System

The VAMP project can viewed as one input to one output. At the head of the project is the voice input by a user that has trained a system. Through various modules, iterations, and network links, VAMP arrives at MIDI output to control music hardware. In between these two steps were our concern, and to faciliate devlopment and outsider understanding of the problem, we split the project into three modules as outlined above. Each module takes input from the previous module and processes it. As shown in the figure above, the spoken voice is the first entry into the system, which

is processed by the Voice Recognition (VR) module. The VR module connects to the parser module by shared code. The parser module takes the discrete words “spit out” by the VR module and parses it into musical meaning. After this parsing is done, the object is sent to the external sequencer through a TCP link. The external sequencer, for the demo purposes of VAMP, indentifies the words and controls an object-oriented MIDI system through calling various methods. The eventual output of the system is MIDI data sent to external music-making equipment.

The first module to tackle was the voice recognition system. Once that was operational, we moved on to the natural language parser, then on to the implementation of the output. This system is efficient for its modularity and its ease of understanding. The approach then is split on each individual module, which are described below in detail.

3.1 Voice Recognition Module

Dragon Naturally Speaking Developer Suite is a Software Developer Kit (SDK) developed by Dragon Systems, Inc to allow software developers to integrate Dragon's voice recognition engine into their own applications.

3.11 Useful Features of SDK

Speech Adaptability

The SDK has the feature of being able to adapt the voice recognition engine to the speaking style of a user. It is able to do this by allowing a user to correct any errors that may arise in the voice recognition. The method of correction is different depending on the application built using the SDK. When a user corrects the voice recognition results the SDK allows the application to be able update the users speech files with new information that will allow the voice recognition engine to recognize the users voice with a higher accuracy.

This feature is useful because it allows the voice recognition of the product to become more reliable with use. With prolonged use, the voice recognition engine will be able to accurately recognize almost everything a user says. This feature allows the voice recognition interface of the application to become a more useful tool.

Multiple Users

The Dragon SDK allows an application to be able to create and maintain multiple users and user settings. This feature is important because of the complexity of recognizing voice input. Every person has a different voice (i.e. accents, slurs, etc.), thus making it very difficult to create a

universal voice recognition program. To get around this problem it is imperative to keep separate speech files for each user of the voice recognition problem. Those speech files can then be configured to more accurately understand the different speaking styles of the individual users.

The usefulness of this feature lies in its ability to make a voice recognition application useful for multiple people. It allows for a variety of people with different speaking styles to be able to user the voice recognition interface easily and more accurately.

System Configuration

Dragon's SDK allowed a voice recognition application to configure the audio components of a users system for use with the voice recognition engine. This feature allows a user to be able to change the audio components of his/her system and quickly setup up the new components for voice recognition. It also allows multiple users to be able to use different audio components, such as microphones, on the same computer.

Programming Language Choice

The SDK allows software developers to develop voice recognition applications in either C++ or Microsoft Visual Basic.

This feature is very useful in the development of a voice recognition application. It gives the developers of the voice recognition application a wider range of programming tools with which to develop their software application.

3.1.2 Problems Encountered

One problem that we found during our use of Dragon's developer suite was that the developer suite itself was not completely implemented. At the time that our application was being developed Dragon had not completely implemented all the features of the SDK. A partial list of the implemented and non-implemented features as according to a help file included with the program can be seen in Appendix B.

An example of this problem is with the feature of the SDK that allows the user to configure his/her audio components for use with the voice recognition engine. Before a user is allowed to run the voice recognition engine he/she must configure his/her audio components. Thus the SDK allows an application to check to see if a user has completed an audio configuration. The problem is that the SDK does not keep track of whether or not the user has run the audio configuration. This problem causes the application to not know that the user has run the audio configuration and therefore not allow the user to access the application.

We were able to overcome this problem by not checking to see if the audio setup had been completed and to instead check to see if the user has been calibrated. For a user to have been calibrated they must first successfully complete an audio setup. So by checking this property we were able to bypass the problem with the audio setup.

3.1.3 User Interface Design

The user interface design is an important part of software development. The user interface is the part of the software that will be seen by the user. Without a well-designed user interface, a useful software product can become cumbersome and lose its usefulness.

Development of Interface

We first looked at what the application was going to be used for. When we did this, we saw that the graphical user interface did not have to be very complex as the majority of the use of the software is through a vocal interface.

The first thing that we needed to design was a user interface that allowed the user to create, select and delete a user profile to be able to access the multiple user feature of the voice recognition engine. We decided to model our interface on the interface that comes with Dragon NaturallySpeaking. We decided on this course of action based on the fact that users tend to learn how to use a program easier if the user interface is similar to others they have used.

Our user interface is show in Figure 2. As seen in the figure the user is shown a box on the left of the screen that displays all currently available speakers. On the right side of the screen the user is given the option of four action buttons. The cancel button, which cancels the selection of the user is placed in a location that is similar to it location in many other software programs. The other three buttons are placed in order of their projected use, with the buttons with the highest projected use on top. The top button is the “Select Speaker” button, which sets the selected speaker as the current speaker and then activates the main form of the program. We projected that this button would get the most use, as a user will normally be selecting a previous user as they start up the program. The next button is the “Create Speaker” button, which activates the create speaker form

to create a new speaker. We projected that this button should be placed underneath the "Select Speaker" button, because each user will use it when they create their speaker profile. The third button, the "Delete Speaker" button, which deletes the selected speaker and all related speech files from the system. We projected that this button would get the least amount of use and by this projected we put as the third action button on the screen.

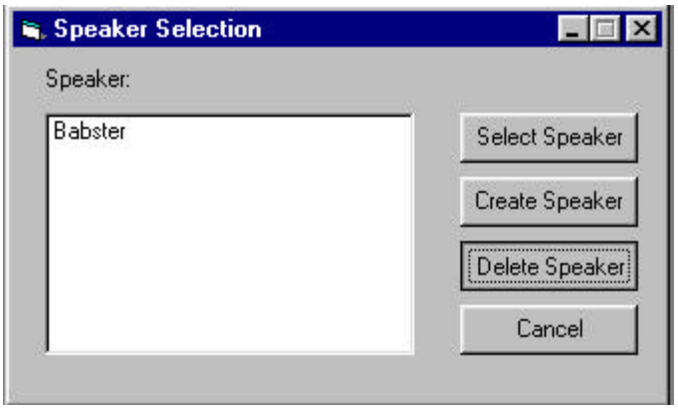

Figure 2 – Screen shot displaying the user interface of the Speaker Selection screen of VAMP

The second thing that we needed to design was the vocal recognition interface for the user. We wanted this interface to be as simple as possible as the user will be using voice input for the majority of the time. So we looked at all the things that a user will need to have control over while they are using our application. When this process was complete we determined that the user would need to have control over the state of the microphone, the output of the voice recognition engine and a manual method of sending the spoken command to our parser for use with the midi sequencer.

As seen can be seen in Figure 3 we decided to use a simple text box to contain the output of the voice recognition engine. Having a simple text box allows the user to be able to see the text output

of the engine and to be able to correct any errors in that may have occurred during the conversion from speech to text.

To allow the user to be able to control the state of the microphone we used a button from Dragon's SDK that managed the microphone state. This button shows the current state of the microphone through the use of changing icons on the button and by displaying a colored intensity bar, which represents the level and intensity of the sound going into the microphone.

We used a simple action button to allow the user to be able to manually run a command that has been recognized by the voice recognition engine and placed in the receiving text box. This button allows a user to be able to shut of the microphone and still be able to run a command. This would be ideal in a high noise situation where there is a lot of external noise that can be picked up by the microphone.

The layout of these three buttons was determined by placing the buttons in different locations and determining which layout was the most aesthetically pleasing. When we completed this determination we were left with the layout as show in Figure 3.

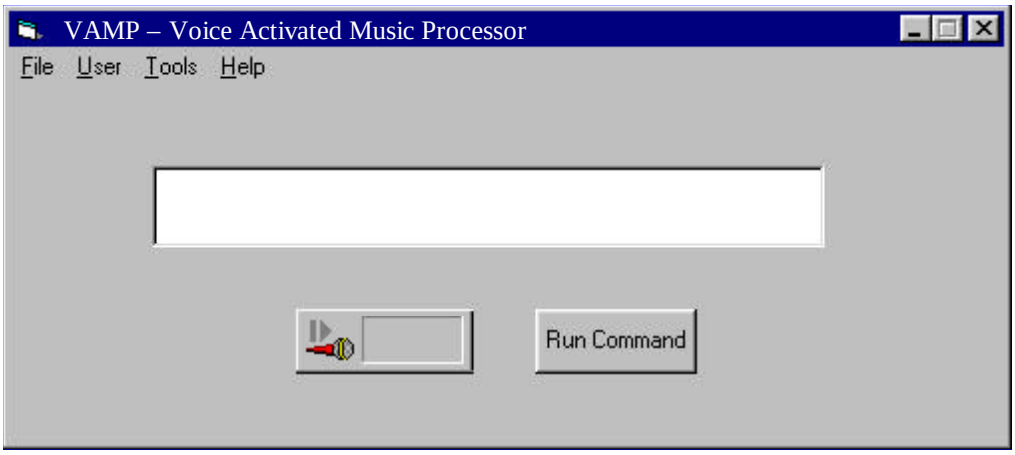

Figure 3 – Screen shot displaying the user interface of the main form of VAMP

The design of the menus, which can be seen in Figure 4, was based on the menu layout in Dragon System's NaturallySpeaking. This decision was also based on the idea of users learning a program quicker if the interface is similar to one they have used before. The similarity in the interfaces is the order of the "File", "User", "Tools" and "Help" menus.

The design of the "File" menu was simple, as the only function it needed to perform was to give the user and option to exit the program. The design of the "User" menu is also taken from the NaturallySpeaking user interface. It contains all of the commands for the maintenance of a user. By using this menu the user is able to create a new speaker profile, open a different speaker profile or to save the speech files of the current speaker profile. The "Tools" menu contains all of the tools that our application allows a user to use. We added the features of being able to setup you audio hardware for voice recognition, to increase voice recognition accuracy by training the voice recognition engine, to edit the command lists that are used by our parser, and to connect to a sequencer through a TCP connection. The order of the "Tools" menu was determined by putting like commands together and by separating the connection tools from the audio/parser tools. The

"Help" configuration was taken from the design of most applications which have help files incorporated into them.

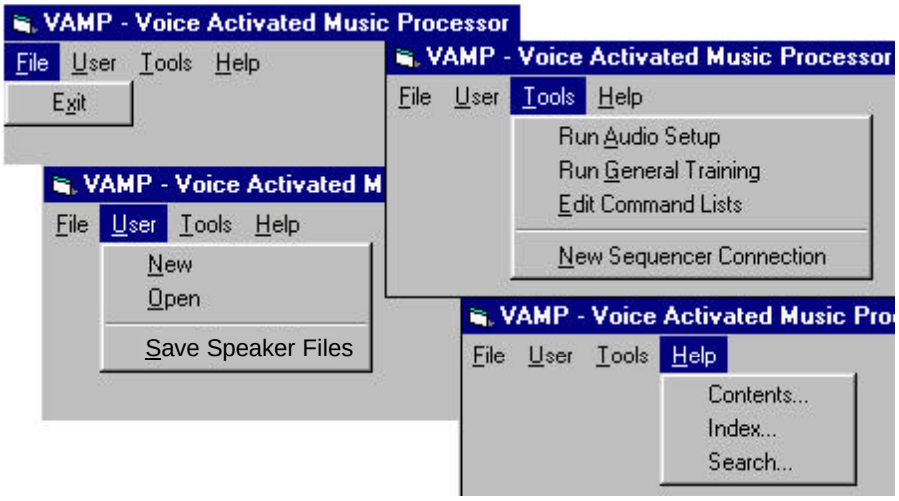

Figure 4 - Screen shot showing the menu layout of the main form of VAMP

Our graphical user interface does not depend on either the Dragon SDK or the form of the parser. We could change voice recognition engines by simple changing the underlying code without modification of the user interface. This ability is also present with the parser. Currently our parser is activated when a user either gives a voice command or when the "Run Command" button is pressed. When called the parser takes the string that is in the voice recognition output text box and parses it. To incorporate a new parser into our system would be a simple task. We would only have to set the input of the new parser to the text in the voice recognition output text box.

3.2 Parser Module

Before we could begin the parser, we first had to determine what grammar we would be allowing and what we would be looking for within that grammar.

In order to formulate the legal grammar, we compiled a list of phrases commonly used in a rehearsal setting. These phrases were collected through interviews, emails and live demonstrations of professional conductors. Once we had the list compiled, we cross-referenced phrases with similar meaning, paying special attention to the wording used.

Through this analysis a distinguishable reoccurring pattern was found. We noticed that there were only a few key words in any given phrase. These key words were then broken down into four categories: Location, Part, Attribute and Action.

The parser's center is a text box. The text box serves as the entrance point for the sentence to be parsed. The text box is important because it allows the user to easily tie the output of the voice recognition into the input of the soon to be parser. As well, if a user did not want to control VAMP with their voice, they could type in the natural language command.

A button is on the form for the user to signal that the sentence was complete and ready to be parsed. When the button is pressed the string is passed to the parser module.

When the parser receives the string, it begins by looking for the word ‘and’. If it should find any, it breaks the sentence into two parts, the “current” part, and the “rest” part. The current part holds the information in the string which precedes the ‘and’. The rest part holds all information after the ‘and’, this section is passed recursively back to the parser until the word ‘and’ is not found.

The next section of the parser works on the current part mentioned above. This part looks for spaces which may exist within a string. When a space is found the string is again broken into two parts, the current part and the rest part. The rest part is passed recursively back to this section of the parser. The current part is passed onto the next phase of the parser.

In the next phase, we check the word received to see if it belongs to one of the four allowed categories. This is accomplished through the use of four lists, one for each category. The word is compared against every word in each list, one at a time. If the word is found, it is immediately placed in a temporary text box for storage, and this section is exited. If the word is not found, the parser continues onto the next word.

Once each word in the sentence is checked against the lists, the parser constructs a command string which will be sent to the sequencer via TCP. This construction is done through the use of four temporary text boxes which are used to hold the words which matched the lists. We simply add one text box to another, placing a colon in between each one. The final string is stored in another text box, ready to be sent to the next section:

```
Location:Part:Attribute:Action

examples:
Measure 3:Violins::Play
:Violins:Volume:Turn Up
```

The other aspect of the parser is the handling of special cases. We found there to be two instances where we needed to receive an argument along with a word. The parser checks for the word ‘measure’ and ‘track’ before the parser compares them to the lists. If one of the words is found, the next word in the sentence is added to it, and they are both stored in the appropriate text box.

3.3 Output

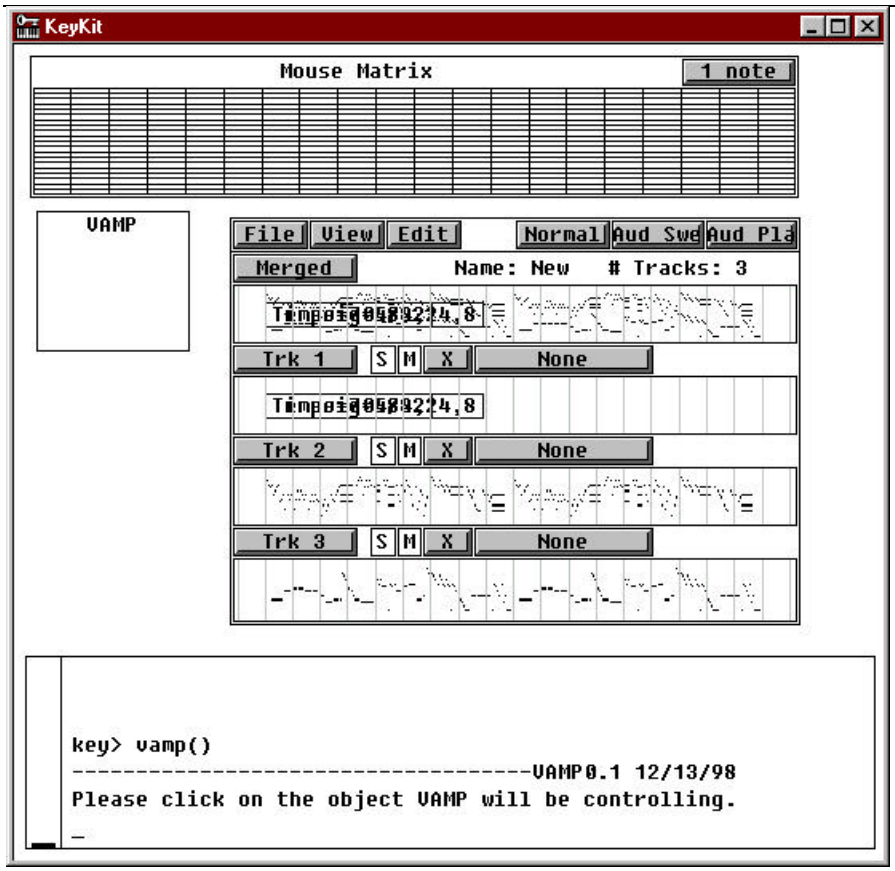

Figure 5 - Screenshot showing VAMP and KeyKit sequencer

VAMP makes use of the publicly available KeyKit (see Figure 7), developed by Tim Thompson in conjunction with AT&T. KeyKit is a cross-platform MIDI development kit that allows for a new language of MIDI signals and allows users to build their own extensions to the software. For prototyping a controllable sequencer, KeyKit proved to be simple and powerful at the same time. For the purposes of VAMP, we both modified KeyKit's sequencer (the middle of Figure 7) and

also created a new tool, called VAMP. (For confusion's sake, we will refer to the KeyKit VAMP tool as VAMP-KeyKit and the parser / voice recognition code as VAMP-VB.)

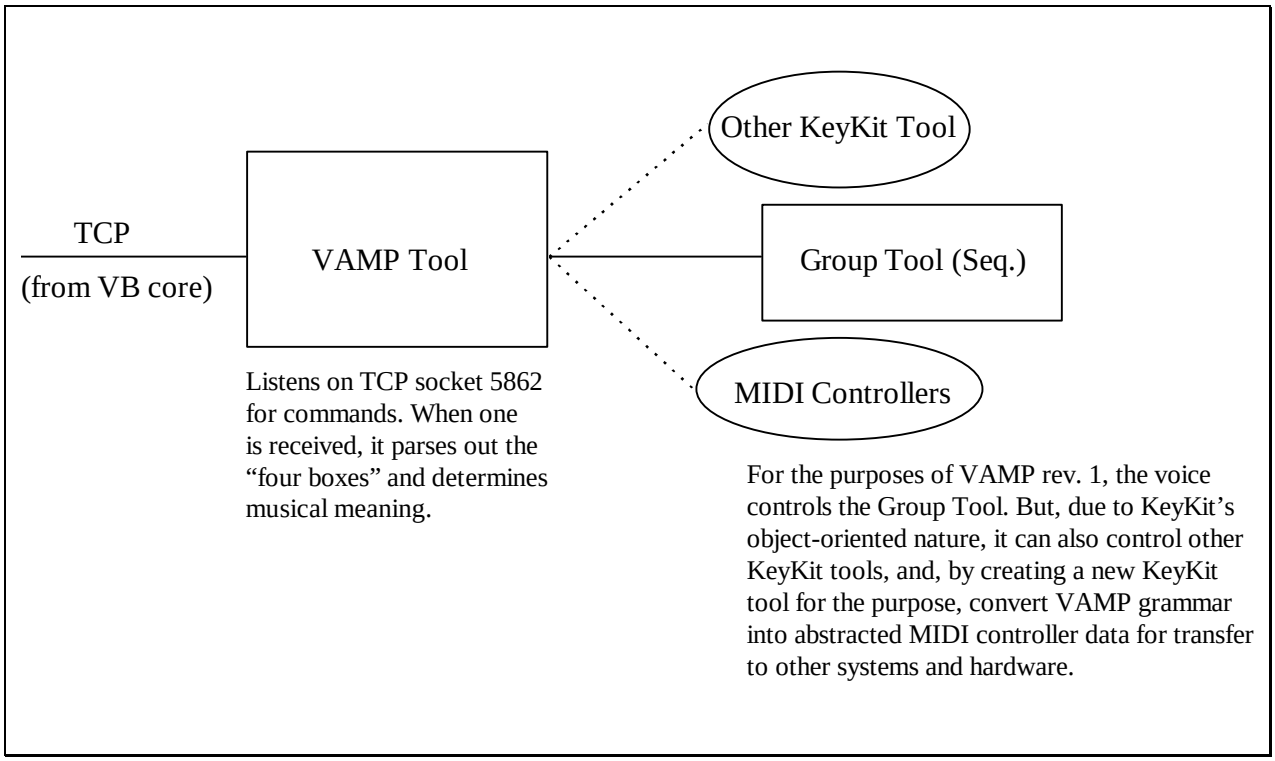

Figure 6 - VAMP-KeyKit Flow Diagram

As shown in Figure 8, VAMP-KeyKit was built as a "tool" in KeyKit to maintain KeyKit's object-based architecture. It listens on a TCP port and then parses out the strings it receives to control another KeyKit object, which for demonstration purposes was a prototyped music sequencer. However, this does not limit the entire VAMP system to control just a sequencer: on the startup of VAMP, it asks the user which object to control; the user merely clicks with their mouse on the KeyKit tool that VAMP will be sending signals to. In the future, different KeyKit objects can be created that do a host of new musical processes, including converting VAMP objects into abstracted MIDI data that other systems and hardware devices can understand.

The sequencer was built on KeyKit's Group tool, a simple yet expansive sequencer. The Group tool can standalone by itself and record MIDI data and play it back as any good sequencer through a series of methods attached to the main Group class. Each method performs an action that can be accessed by mouse control or through other tools, such as VAMP. The Group tool was enhanced with new VAMP-specific methods for the purposes of the project, including seeking to a particular measure, muting one particular track, and slowing down the tempo.

When the VAMP-KeyKit tool is invoked (either through KeyKit's menu structure or by calling a vamp() command from the Console window), it asks the user to select a controlled tool. At this point a user should have the tool open that they wish to have controlled by voice. The tool is selected by clicking on it with the mouse, which calls back the VAMP tool with a pointer to the object. This pointer, which is referenced as **tool** throughout the VAMP-KeyKit code, is necessary to access the object's methods: for example, VAMP can now simply tell the sequencer to play by invoking **tool.playaudition(time1,time2)**.

After selecting the destination tool, VAMP sets up its server to listen on port 5862 on the system's TCP stack. From this point on, VAMP-KeyKit and VAMP-VB are connected through a TCP networking link. TCP is a novel solution due to its ease of sending buffered strings across networks: since we have to wait for the entire string to compose itself from the VR system, the "buffer lag time" is not an issue. While all of our testing was done with both VAMP-VB and VAMP-KeyKit on the same machine, there is no limit to the distance of the two modules. VAMP-

KeyKit can be on any computer connected to the Internet while VAMP-VB can be on a different machine in the same room or thousands of miles away. When a connection is detected on port 5862, VAMP then begins waiting for a message.

A message is defined by VAMP-KeyKit as a string delimited by colons (:), followed by a carriage return and line feed. This conforms to what the VAMP-VB sends the parsed message as. Every time a command is issued from VAMP-VB, the KeyKit code interprets it and immediately breaks it down again into the 'four boxes.' Multiple commands are processed in the order that they are received. Upon receipt of a command and its subsequent parsing, VAMP begins its work:

```
Psuedocode for VAMP-KeyKit's musical meaning parser:

if action is "play"
      if part
            tool.solo(part)
      if location
            tool.setaudition(converttobeats(location), end)
      tool.playaudition
. . .
```

The first revision of the VAMP system has a limited 'musical meaning dictionary.' It can play or stop a sequence, mute or solo instruments, raise tempo or volume, and start at different measures. Future revisions to the VAMP system can easily add functionality by modifying the VAMP-KeyKit code, or developing another solution to receive VAMP-VB's TCP messages. The reason for this modularity was that, at the time of this writing, the better sequencers and music control software appeared for Macintosh systems only, and the better voice recognition packages (i.e., Dragon) were

only for Windows systems. A future application of VAMP would be to have the musical meaning parser on the Macintosh platform while the voice recognition occurs on the Windows platform. The two computers can then be simply linked over Ethernet.

KeyKit's object-oriented nature allows for another advantage: these commands can occur in real time, as a sequence is playing. The TCP-listener and VAMP module are set up as KeyKit 'tasks', which are analogous to UNIX's processes. A programmer can create or delete processes at will, and they run until they are told to delete themselves. This ensures that the possibilities for musical meaning parsing are endless.

4. Evaluation

In our final testing, VAMP performs as promised, with few minor issues. As described in the Voice Recognition module section, the Dragon SDK has implementation errors that do not allow it to retain the user's audio setup from session to session. At the time of this writing, we are in contact with Dragon to work on a fix. Otherwise, a properly trained test subject claimed a high rate of accuracy from Dragon and therefore a good deal of success in controlling VAMP. The subset of grammar implemented in VAMP musically performed well: we are able to control starting, stopping, raising volume, muting and "soloing" (only playing) tracks, and also more detailed editing operations: raising volume by measure, raising tempo, etc.

The connections between the three modules work flawlessly. Over a local system, where the VAMP-VB and VAMP-KeyKit modules converse over TCP on port 5862, there is only lag from the VR system determining speech. We estimate on average a lag time of 1-2 seconds from finished spoken phrase to KeyKit's response. This conforms to our feasibility research, in which we read that a similar system (in which doctors were controlling an instructional surgery video from "the floor") had slightly longer lag periods, due to slower hardware at the time. As computers only get faster, we expect this lag time to minimize to unsubstantial.

The accuracy of the VR system, while close to 90 percent, can be enhanced. We will discuss possible future solutions below. Many of the mistakes that the system made, however, are parsed

out by the module and not sent to KeyKit. For the time being, then, misheard phrases have no effect on the music.

In our view, the project is a success: an ultimately expandable framework for voice control of music. Our demonstration proves the feasibility of such a task, while in the future we hope to see many enhancements to turn VAMP into a practical and seamless connection between voice and music.

5. Conclusions

As voice control becomes more ordinary to ordinary users, and developers are realizing that voice is the most natural way to communicate, projects such as VAMP will become more prominent. One view of the future of computing places small intelligent voice-activated devices to handle different tasks, such as word processing, communications, development, and the arts. Each device can be geared towards their particular function, since a task such as VAMP's is extraordinarily different from a simple 'dictation' package.

By developing musical grammar and implementing it in a practical situation, VAMP creates a new paradigm for music control based on natural language. The voice connection only enhances this: and while voice recognition can only get accurate, the number of users implementing voice recognition in their day-to-day activities can only increase. 'Expert' users that require optimal control of their systems need devices that accurately control their work (and art) will soon desire products like VAMP, products that "understand them" without any need for complication.

VAMP's modularity and expandability are its strong points: it would be foolish to assume that this incarnation is the final one, and by allowing for hooks into all points of the process, it ensures that it can grow with the technology. VAMP's promise is far-reaching and pertinent, especially as music systems and the computers that control them only get more complicated.

6. Future Work

By laying the framework for a speech to musical object solution, the field is now open for a vast number of improvements and implementations. The VAMP core and current state allow it to control a limited set of musical parameters, but it is ultimately expandable, both within the code and through its Parser module. By adding new words to the vocabulary of the Parser, a new musical command can arise, given that the programmer would also know KeyKit's (or whatever the output phase would be at that point) language to physically implement the command.

The placement of KeyKit in our project was simply for rapid development and prototyping purposes. While KeyKit is a powerful and widely-supported program, it might not suit the needs of all future users. The modularity of the system exists so that a new developer can replace the KeyKit module with another MIDI solution, either off-the-shelf of self-coded. Opcode's Max is a more robust and better-supported solution, although it is only available for the Macintosh platform. However, with the TCP link, a future group can enhance the project to include Max running on a Macintosh while the VR module runs on a dedicated Windows machine. This can also enhance the 'intelligence' of VAMP: if the output module understands the user's hardware (for example, if their sound module can receive filter changes on channel 4 that make a sound 'brighter') then it can make those adjustments directly to the hardware, giving a user even more complete control.

Other advancements can be made in the VR section of the project; it will accept any Microsoft SAPI-compliant package, and as the technology and accuracy of these VR programs increase, the 'old' Dragon core in place now can be swapped out with little effort. To increase accuracy, a

developer can install a hardware device on the microphone to eliminate stray signals, or install a noise-canceling device to eliminate background sounds interfering.

The parser can be made more robust, other than just the vocabulary: by keeping a buffer of voiced commands, VAMP can ‘estimate’ what the user is trying to get across. For example, a user that said “Make the violins louder” could then next say, “No, louder.” and VAMP would know to check the previous statement. This would involve a small amount of ‘pseudo-intelligence,’ or could be implemented from a publicly-available natural language parser.

References

Chadabe, Joel. Electric Sound: The Past and Promise of Electronic Music. New York: Prentice Hall, 1996.

Cui, Weylou, et al. "Voice-Aware Support for Multimedia Applications." Integrated Media Systems Center, University of Southern California.

Dragon Systems, Inc. corporate web site: http://www.dragonsystems.com

Kay, Russell. "Do You Hear What I Say?" Byte, January 1998, pp 115-116.

Manes, Stephen. "Speech Recognition, Now You're Talking!" PC World, October 1997. Web resource: http://www2.pcworld.com/software/utility/articles/oct97/1510p400.html

Puckette, Miller. "Pure Data: Another Integrated Computer Music Environment." Proceedings, Second Intercollege Computer Music Conference, Tachikawa, pp 37-41.

Rowe, Robert. Interactive Music Systems: Machine Listening and Composing. Cambrdige: MIT Press, 1993.

Schmandt, Christopher. Voice Communication With Computers: Conversational Systems. New York: Van Hostrand Reinhold, 1994.

Smith, Ronnie. “ An evaluation of strategies for selectively verifying utterance meanings in spoken natural language dialog.” International Journal of Human-Computer Studies, 48, pp 627-647.

Sudkamp, Thomas. Languages and Machines: Second Edition. Reading: Addison-Wesley, 1998.

Appendix A: Platform

The VAMP executable needs to run in a Windows 95 / 98 / NT system with Dragon NaturallySpeaking installed. The VAMP installer takes care of all the support software you need, and will attempt to detect the presence of NaturallySpeaking, letting the user know they need to install it if it is not found. The VAMP installer also installs the following pieces of software:

1. The Parser / front end 'core' (Windows executable)
2. AT&T's KeyKit with supporting VAMP-hooks
3. NaturallySpeaking developer hooks and training documentation

Upon starting VAMP for the first time, a user needs to train Dragon if they have not already done so. If VAMP is being installed on a system different from the one the user has trained Dragon, the user needs to move their 'voice print' over from the old system. This is accomplished by merely copying the applicable directory over to the new system.

KeyKit requires that a MIDI-capable sound card be installed. This sound card can use its own internal General MIDI voices or be configured through Windows' MIDI Mapper to control various outboard synthesizers and sound modules. KeyKit's included documentation has more on this topic.

It is possible that the system containing KeyKit and / or the sound modules and the VAMP system be on separate machines, due to the TCP nature of the link between the two modules. This is realized by entering in a different value than 'Localhost' in the TCP setup properties of the VAMP

software. KeyKit on the ‘listener’ machine does not need to be modified, it merely waits for a message from any IP address and acts on it.

Appendix B: List of Implemented Interfaces

List of Implemented/Not Implemented Interfaces

Voice Command API (VCmd)

The Voice Command API (VCmd) allows users to control an application by speaking commands through an audio input device, rather than by using the mouse or keyboard.

IVCmdAttributes:	**Implemented**
AutoGainEnableGet	Implemented
AutoGainEnableSet	Implemented
AwakeStateGet	Implemented
AwakeStateSet	Implemented
DeviceGet	Implemented
DeviceSet	Implemented
EnabledGet	Implemented
EnabledSet	Implemented
MicrophoneGet	Implemented
MicrophoneSet	Implemented
SpeakerGet	Implemented
SpeakerSet	Implemented
SRModeGet	Implemented
SRModeSet	Implemented with caveat(s): Can't set to current mode ID.
ThresholdGet	Implemented
ThresholdSet	Implemented

IVCmdDialogs:	**Implemented**
AboutDlg	Implemented
GeneralDlg	Implemented with caveat(s): This method hangs with Dragon NaturallySpeaking version 3.01.
LexiconDlg	Not implemented
TrainGeneralDlg	Not implemented
TrainMicDlg	Not implemented

IVCmdEnum:	**Implemented**
Clone	Implemented
Next	Implemented
Reset	Implemented
Skip	Implemented

IVCmdMenu:	**Implemented**
Add	Implemented
Deactivate	Implemented
EnableItem	Implemented
Get	Implemented
ListSet	Implemented
ListGet	Implemented
Num	Implemented
Set	Implemented

SetItem	Implemented
TrainMenuDlg	Not implemented
Activate	Implemented
Remove	Implemented

IVoiceCmd:	**Implemented**
CmdMimic	Implemented
MenuCreate	Implemented
MenuDelete	Implemented
MenuEnum	Implemented
Register	Implemented

IVCmdNotifySink:	**Implemented**
AttribChanged	Implemented
CommandOther	Implemented
CommandRecognize	Implemented
CommandStart	Implemented
Interference	Not implemented
MenuActivate	Implemented
UtteranceBegin	Implemented
UtteranceEnd	Implemented
VUMeter	Not implemented

Voice Text API (VTxt)

The Voice Text (VTxt) API provides simple text-to-speech (TTS) capabilities for applications.

Note: The **Italian** version of Dragon NaturallySpeaking does not provide a SAPI-compliant TTS engine. Developers wishing to use the VTxt interfaces for Italian, must use a TTS engine other than the one provided with NaturallySpeaking.

IVTxtAttributes:	**Implemented**
DeviceGet	Implemented
DeviceSet	Implemented
EnabledGet	Implemented
EnabledSet	Implemented
IsSpeaking	Implemented
SpeedGet	Implemented
SpeedSet	Implemented
TTSModeGet	Implemented
TTSModeSet	Implemented

IVTxtDialogs:	**Implemented**
AboutDlg	Implemented
GeneralDlg	Not implemented
LexiconDlg	Not implemented
TranslateDlg	Not implemented

IVTxtNotifySink:	**Implemented**
AttribChanged	Implemented
SpeakingDone	Implemented
Speak	Implemented
SpeakingStarted	Implemented
Visual	Implemented

IVoiceText:	**Implemented**
Register	Implemented
Speak	Implemented with caveat(s): ("") returns: 0x8007000E
StopSpeaking	Implemented
AudioFastForward	Implemented
AudioPause	Implemented
AudioResume	Implemented
AudioRewind	Implemented

Speech Recognition API (SR)

ISRCentral:	**Implemented**
GrammarLoad	Implemented
ModeGet	Implemented
Pause	Implemented
PosnGet	Implemented
Register	Implemented
Resume	Implemented
ToFileTime	Implemented
UnRegister	Implemented

ISRDialogs:	**Implemented**
AboutDlg	Implemented
GeneralDlg	Implemented with caveat(s): This method hangs with Dragon NaturallySpeaking version 3.01.
LexiconDlg	Not implemented
TrainMicDlg	Not implemented
TrainGeneralDlg	Not implemented

ISREnum:	**Implemented**
Clone	Implemented
Next	Implemented
Reset	Implemented
Select	Implemented
Skip	Implemented

ISRFind:	**Not implemented**
Find	Not Implemented
Select	Not Implemented

ISRGramCFG:	**Implemented**
LinkQuery	Not implemented
ListAppend	Implemented
ListGet	Implemented
ListQuery	Implemented
ListRemove	Implemented
ListSet	Implemented

ISRGramCommon:	**Implemented**
Activate	Implemented
Archive	Implemented
BookMark	Not implemented
Deactivate	Implemented

DeteriorationGet	Implemented
DeteriorationSet	Implemented
TrainDlg	Not implemented
TrainPhrase	Not implemented
TrainQuery	Not implemented
ISRGramDictation:	**Implemented**
Context	Implemented
Hint	Not implemented
Words	Implemented
ISRGramInsertionGUI:	**Implemented**
Hide	Implemented
Move	Implemented
Show	Implemented
ISRResAudio:	**Implemented**
GetWAV	Implemented
ISRResBasic:	**Implemented**
FlagsGet	Not implemented
Identify	Not implemented
PhraseGet	Implemented
TimeGet	Implemented with caveat(s): (NULL,NULL): expected E_INVALIDARG, got E_UNEXPECTED
ISRResCorrection:	**Implemented**
Correction	Implemented
Validate	Implemented
ISRResEval:	**Implemented**
ReEvaluate	Not implemented
ISRResGraph:	**Implemented**
BestPathPhoneme	Not implemented
BestPathWord	Implemented
GetPhonemeNode	Not implemented
GetWordNode	Implemented with caveat(s): (dwWrdNde,&wrdnde,NULL,0,NULL): expected E_INVALIDARG, got S_OK
PathScorePhoneme	Not implemented
PathScoreWord	Not implemented
ISRResMemory:	**Implemented**
Free	Implemented
LockGet	Implemented
LockSet	Implemented
Get	Not implemented
ISRResMerge:	**Implemented**
Merge	Not implemented
Split	Implemented
ISRResModifyGUI:	**Not implemented**
Hide	Not Implemented

Move	Not Implemented
Show	Not Implemented
ISRResScores:	**Not implemented**
GetPhraseScore	Not Implemented
GetWordScores	Not Implemented
ISRResSpeaker:	**Not implemented**
Correction	Not Implemented
Identify	Not Implemented
IdentifyForFree	Not Implemented
Validate	Not Implemented
ISRSpeaker:	**Implemented**
Delete	Implemented
Enum	Implemented
Merge	Not implemented
New	Implemented
Query	Implemented with caveat(s): (szSpkr, 0, NULL): expected E_INVALIDARG, got S_OK
Read	Not implemented
Revert	Not implemented
Select	Implemented
Write	Not implemented
ISRAttributes:	**Implemented**
AutoGainEnableGet	Implemented
AutoGainEnableSet	Implemented
EchoGet	Implemented
EchoSet	Implemented
EnergyFloorGet	Implemented
EnergyFloorSet	Implemented
MicrophoneGet	Implemented with caveat(s): (NULL,0,NULL): expected E_INVALIDARG, got S_OK
MicrophoneSet	Implemented
RealTimeGet	Implemented
RealTimeSet	Implemented
SpeakerGet	Implemented with caveat(s): (NULL,0,NULL): expected E_INVALIDARG, got S_OK
SpeakerSet	Implemented with caveat(s): (NULL): expected E_INVALIDARG, got E_UNEXPECTED
ThresholdGet	Implemented
ThresholdSet	Implemented
TimeOutGet	Implemented
TimeOutSet	Implemented
ILexPronounce:	**Implemented**
Add	Implemented
Get	Implemented
Remove	Implemented
ISRGramNotifySink:	**Implemented**
BookMark	Not implemented
Paused	Implemented
PhraseFinish	Implemented

PhraseHypothesis	Not implemented
PhraseStart	Implemented
ReEvaluate	Not implemented
Training	Not implemented
UnArchive	Not implemented
ISRNotifySink:	**Implemented**
AttribChanged	Implemented
Interference	Not implemented
Sound	Not implemented
UtteranceBegin	Implemented
UtteranceEnd	Implemented
VUMeter	Not implemented

Appendix C: Grammar

We arranged our acceptable phrases into a generic grammar for ease of parsing. What follows is a regular expression list of the grammar followed by two examples of each type.

[Instrument/Track]*[Less/More]*[Dynamic/Expression]*[Location [to Location]]
Trumpets need to be louder at measure 12.
Violins need to be less staccato.
[Location [to Location]]*[Less/More]*[Dynamic/Expression]*[Instrument/Track]
At the Coda I want more from the saxophones.
At Measure 3 I want more emphasis on the downbeat.
[Location [to Location]]*[Instrument/Track]*[Less/More]*[Dynamic/Expression]
At measure 5 I want the cellos to play more passionately.
At the second ending I want the Tubas to play very gently.
[Location Directive]*[Location]
Start at Measure 8.
Take if from the top.
* The asterisk represents any unrecognized words or phrases.

Following are example words for each of the groups that the parser can understand.
[Instrument/Track]
Track # [1,2,3,4…]
Cello
Violin
Viola
Bass
Trumpet
Tuba
Trombone
Etc.

[Dynamic/Expression]
Slow
Volume
Pianissimo
Piano
Mezzo Piano
Mezzo Forte
Forte
Fortissimo
Sforzando
Agitati
Anima Soul
Animato
Apassionato
Passionately

Brilliant
Sadly
Playful
Etc.

[Less/More]
- Less
- More
- Not
- Very
- Don’t

[Location]
- Measure # [1,2,3,4…]
- Top of the page
- Top
- Edge
- Andante
- Allegro
- Beginning
- Coda
- Segno
- Largo
- Lento
- Adagio
- Solo
- Entrance
- Page # [1,2,3,4…]
- Etc.

[Location Directive / Action]
- Start
- Go
- From
- Play
- Etc.

Examples:

Trumpets need to be louder at measure 12.
Violins need to be less staccato.
Get Louder at the tutti section.
Not so loud.
Don't use so much vibrato.
At the Coda I want more from the saxophones.

At Measure 3 I want more emphasis on the downbeat.
At measure 5 I want the cellos to play more passionately.
At the second ending I want the Tubas to play very gently.
Start at Measure 8.
Take if from the top.
Go Back to the top of the page.
From the top
Softer.
Louder.
Make the staccatos shorter.
Accents need to be louder.
Really decrescendo there.
Take it from the second ending.
We need more vibrato from the trumpets.
Make the crescendo stand out.
Not so loud clarinets.
More vibrato.
Don't use vibrato here, make it quite simple.
I need a brighter sound from the brass.

The scope of each verbal command is based upon the arguments received. If a conductor were to say “Track 1 more volume”, then the increase in volume would be applied only to Track 1. However, a conductor is allowed to say “LOUDER!” which would cause an increase in volume in all tracks.

Any command affecting tempo would have to be applied to all tracks. It would not make sense to slow down one instrument while maintaining the other’s speeds.

Appendix D: Source Code

Following is the Visual Basic source code for the VAMP-VB modules.

Figure 7 - Main Form Source Code

```
VERSION 5.00

Object = "{5C486340-2F92-11D1-A47C-00A024A3A678}#1.0#0"; "DNSTK10.DLL"

Object = "{33101C00-75C3-11CF-A8A0-444553540000}#1.0#0"; "CSWSK32.OCX"
Begin VB.Form frmmain
   Caption         =   "VAMP - Voice Activated Music Processor"
   ClientHeight    =   2685
   ClientLeft      =   165
   ClientTop       =   450
   ClientWidth     =   7485
   LinkTopic       =   "Form1"
   MaxButton       =   0   'False
   ScaleHeight     =   2685
   ScaleWidth      =   7485
   StartUpPosition =   1  'CenterOwner
   Begin SocketWrenchCtrl.Socket Socket1
      Left            =   480
      Top             =   720
      _Version        =   65536
      _ExtentX        =   741
      _ExtentY        =   741
      _StockProps     =   0
      AutoResolve     =   -1  'True
      Backlog         =   5
      Binary          =   -1  'True
      Blocking        =   -1  'True
      Broadcast       =   0   'False
      BufferSize      =   0
      HostAddress     =   ""
      HostFile        =   ""
      HostName        =   ""
      InLine          =   0   'False
      Interval        =   0
      KeepAlive       =   0   'False
      Library         =   ""
      Linger          =   0
      LocalPort       =   0
      LocalService    =   ""
      Protocol        =   0
      RemotePort      =   0
      RemoteService   =   ""
      ReuseAddress    =   0   'False
      Route           =   -1  'True
      Timeout         =   0
      Type            =   1
      Urgent          =   0   'False
   End
   Begin VB.TextBox txtAct
      Height          =   285
```

```
   Left            =   4920
   TabIndex        =   11
   Top             =   240
   Visible         =   0   'False
   Width           =   1095
End
Begin VB.TextBox txtAtt
   Height          =   285
   Left            =   3720
   TabIndex        =   10
   Top             =   240
   Visible         =   0   'False
   Width           =   1095
End
Begin VB.TextBox txtPar
   Height          =   285
   Left            =   2400
   TabIndex        =   9
   Top             =   240
   Visible         =   0   'False
   Width           =   1095
End
Begin VB.TextBox txtLoc
   Height          =   285
   Left            =   1200
   TabIndex        =   8
   Top             =   240
   Visible         =   0   'False
   Width           =   1095
End
Begin VB.TextBox txtCom
   Height          =   285
   Left            =   1320
   TabIndex        =   7
   Top             =   2280
   Visible         =   0   'False
   Width           =   4575
End
Begin VB.CommandButton cmdRun
   Caption         =   "Run Command"
   Height          =   495
   Left            =   3960
   TabIndex        =   6
   Top             =   1680
   Width           =   1215
End
Begin VB.ListBox List4
   Height          =   255
   Left            =   4800
   TabIndex        =   5
   Top             =   1320
   Visible         =   0   'False
   Width           =   1215
End
Begin VB.ListBox List3
   Height          =   255
   Left            =   3600
   TabIndex        =   4
   Top             =   1320
   Visible         =   0   'False
   Width           =   1215
End
Begin VB.ListBox List2
   Height          =   255
```

```
   Left            =   2400
   TabIndex        =   3
   Top             =   1320
   Visible         =   0   'False
   Width           =   1215
End
Begin VB.ListBox List1
   Height          =   255
   Left            =   1200
   TabIndex        =   2
   Top             =   1320
   Visible         =   0   'False
   Width           =   1215
End
Begin DNSToolsCtl.DgnDictEdit DgnDictEdit1
   Left            =   6240
   OleObjectBlob   =   "main.frx":0000
   Top             =   480
End
Begin DNSToolsCtl.DgnVoiceCmd DgnVoiceCmd1
   Left            =   6240
   OleObjectBlob   =   "main.frx":0028
   Top             =   720
End
Begin VB.TextBox txtVREntry
   Height          =   615
   Left            =   1080
   TabIndex        =   1
   Top             =   600
   Width           =   5055
End
Begin DNSToolsCtl.DgnMicBtn DgnMicBtn1
   Height          =   495
   Left            =   2160
   OleObjectBlob   =   "main.frx":004C
   TabIndex        =   0
   Top             =   1680
   Width           =   1335
End
Begin DNSToolsCtl.DgnEngineControl DgnEngineControl1
   Left            =   6240
   OleObjectBlob   =   "main.frx":007C
   Top             =   960
End
Begin VB.Menu mnuFile
   Caption         =   "&File"
   Begin VB.Menu smnuExit
      Caption         =   "E&xit"
   End
End
Begin VB.Menu mnuUser
   Caption         =   "&User"
   Begin VB.Menu mnuCreate
      Caption         =   "&New"
   End
   Begin VB.Menu mnuSelect
      Caption         =   "&Open"
   End
   Begin VB.Menu mnuBlank
      Caption         =   "-"
   End
   Begin VB.Menu mnuSave
      Caption         =   "&Save Speach Files"
   End
```

```
   End
   Begin VB.Menu mnuTools
      Caption         =   "&Tools"
      Begin VB.Menu mnuAudio
         Caption         =   "Run &Audio Setup"
      End
      Begin VB.Menu mnuGenTrain
         Caption         =   "Run &General Training"
      End
      Begin VB.Menu mnuEditLists
         Caption         =   "&Edit Command Lists"
      End
      Begin VB.Menu mnuBlank2
         Caption         =   "-"
      End
      Begin VB.Menu mnuNewConnection
         Caption         =   "&New Sequencer Connection"
      End
   End
   Begin VB.Menu mnuHelp
      Caption         =   "&Help"
      Begin VB.Menu mnuContents
         Caption         =   "Contents..."
      End
      Begin VB.Menu mnuIndex
         Caption         =   "Index..."
      End
      Begin VB.Menu mnuSearch
         Caption         =   "Search..."
      End
   End
End
Attribute VB_Name = "frmmain"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Compare Text

Dim VMenu As IVMenuAuto                     'voice recognition command menu
Dim activated As Boolean                    'hold if form has been activated

Const wake_up = 1                           'command numbers
Const go_sleep = 2
Const run_com = 3

'---------------------------------------------------------------------------
'
'Function:  Activate
'
'Purpose:   Sets up form and all variables
'
'---------------------------------------------------------------------------

Public Sub Activate()

    Dim fnum As Integer
    Dim txt As String
    Dim Host As String

    Me.Enabled = True

    List1.Clear                             'clear the list
```

```
On Error GoTo ErrorHandler
fnum = FreeFile
Open App.Path & "\List1.dat" For Input As fnum
                                    'open a file for reading
Do While Not EOF(fnum)              'go through the entire file
    Line Input #fnum, txt           'read a line of the file
    txt = Trim$(txt)                'set txt equal to the line
    If Len(txt) > 0 Then List1.AddItem txt
Loop                                'if txt exists then add it
                                    'to the list

Close fnum

List2.Clear                         'clear the list

fnum = FreeFile
Open App.Path & "\List2.dat" For Input As fnum
                                    'open a file for reading
Do While Not EOF(fnum)              'go through the entire file
    Line Input #fnum, txt           'read a line of the file
    txt = Trim$(txt)                'set txt equal to the line
    If Len(txt) > 0 Then List2.AddItem txt
Loop                                'if txt exists then add it
                                    'to the list
Close fnum

List3.Clear                         'clear the list

fnum = FreeFile
Open App.Path & "\List3.dat" For Input As fnum
                                    'open a file for reading
Do While Not EOF(fnum)              'go through the entire file
    Line Input #fnum, txt           'read a line of the file
    txt = Trim$(txt)                'set txt equal to the line
    If Len(txt) > 0 Then List3.AddItem txt
Loop                                'if txt exists then add it
                                    'to the list
Close fnum

List4.Clear                         'clear the list


fnum = FreeFile
Open App.Path & "\List4.dat" For Input As fnum
                                    'open a file for reading
Do While Not EOF(fnum)              'go through the entire file
    Line Input #fnum, txt           'read a line of the file
    txt = Trim$(txt)                'set txt equal to the line
    If Len(txt) > 0 Then List4.AddItem txt
Loop                                'if txt exists then add it
                                    'to the list

Close fnum


frmWait.Caption = "Creating Form"       'display wait dialog
frmWait.lblWait.Caption = "Creating Form.     Please Wait."
Me.Show



Me.Enabled = False



frmWait.Show
```

```
        activated = True

        DgnEngineControl1.Register              'register voice recognition objects
        DgnMicBtn1.Register
        DgnVoiceCmd1.Register ""
        DgnDictEdit1.Register txtVREntry.hWnd

                                                'create and setup voice recognition command menu
        Set VMenu = DgnVoiceCmd1.MenuCreate("app", "menu", dgnlangUSEnglish, "", vcmdmc_CREATE_TEMP)

        VMenu.Add wake_up, WakeUp, "", ""
        VMenu.Add go_sleep, GoToSleep, "", ""
        VMenu.Add run_com, "run command", "", ""

        VMenu.hWndMenu = hWnd
        VMenu.Active = True

        Unload frmWait

        Me.Enabled = True
        Me.SetFocus

        DgnMicBtn1.Enabled = True               'initialize microphone
        DgnMicBtn1.MicState = dgnmicOff

        txtVREntry.SetFocus

        GoTo Quit

    ErrorHandler:

        If Err.Number = E_NOTIMPL Then          'checks to see if error was caused by non-support of
    dictation

            MsgBox "Your speach engine does not support dictation. Please install one with these
    capabilites.", vbOKOnly, "Error"
            Unload frmWait
            Unload Me

        Else

                                                'displays general error message and exits
            MsgBox Str(Err) + " - " + Error$, vbOKOnly, "Error" ' show message
            Unload frmWait
            Unload Me

        End If

    Quit:


        Socket1.AddressFamily = AF_INET         'sets up socket
        Socket1.Binary = False
        Socket1.Blocking = False
        Socket1.BufferSize = 1024
        Socket1.Protocol = IPPROTO_IP
        Socket1.SocketType = SOCK_STREAM



        Socket1.RemoteService = "echo"



        Socket1.HostFile = ""

        Socket1.HostName = "localhost"          'sets up defaul connection
        Socket1.RemotePort = 5862
        Socket1.LocalPort = IPPORT_ANY
```

```
    If Socket1.Connect <> 0 Then Exit Sub   'checks if connected


End Sub

'---------------------------------------------------------------------------
'
'Function:  cmdRun_Click
'
'Purpose:   Manually runs a command
'
'---------------------------------------------------------------------------

Private Sub cmdRun_Click()
    Dim AndStuff As ParseType


    AndStuff.Current = ""               'initialize the andstuff var
    AndStuff.LeftOver = txtVREntry
    Call ParseControl(AndStuff)         'call the parser with andstuff

    txtVREntry.SetFocus
    txtVREntry.Text = ""

End Sub

'---------------------------------------------------------------------------
'
'Function:  DgnEnginecontrol1_DialogClosed
'
'Purpose:   Is called when a DgnEngineControl function exits.
'
'Variables: Dialog   - specifies which dialog has exited
'           ExitCode - hold exit status of dialog
'
'---------------------------------------------------------------------------

Private Sub DgnEngineControl1_DialogClosed(Dialog As DNSToolsCtl.DgnDialogConstants, ExitCode As
Long)

                                    'check if general training has exited
    If Dialog = dgndlgGeneralTraining Then

                                    'display waiting dialog
        frmWait.Caption = "Saving User"
        frmWait.lblWait.Caption = "Saving " + frmWait.lblWait.Caption
        frmWait.Show

                                    'save speaker
        DgnEngineControl1.SpeakerSave

        Unload frmWait

    End If

    Me.Enabled = True
    Me.SetFocus




End Sub

'---------------------------------------------------------------------------
'
'Function:  DgnVoiceCmd1_CommandRecognize
```

```
'Purpose:   Executes the command that is recognized by the voice recognition
'           engine.
'
'Variables: Command     - name of command recognized
'           ID          - integer id of recognized command
'           Action      - contains string of action to be performed
'           ListResults - results of voice recognition
'
'--------------------------------------------------------------------------

Private Sub DgnVoiceCmd1_CommandRecognize(Command As String, ID As Long, Action As String,
ListResults As DNSToolsCtl.DgnStrings)
   Dim AndStuff As ParseType

    If ID = wake_up Then            'check if command is microphone wake up command

        DgnMicBtn1.MicState = dgnmicOn

    Else

        If ID = go_sleep Then       'check if command is microphone sleep command

            DgnMicBtn1.MicState = dgnmicSleeping

        Else

            If ID = run_com Then    'check if command is run command command

                AndStuff.Current = ""           'initialize the andstuff var
                AndStuff.LeftOver = txtVREntry
                Call ParseControl(AndStuff)     'Call the parser with andstuff

                txtVREntry.SetFocus
                txtVREntry.Text = ""

            End If

        End If

    End If

End Sub

'--------------------------------------------------------------------------
'
'Function:  Form_Load
'
'Purpose:   Initializes variables and calls select speaker form
'
'--------------------------------------------------------------------------

Private Sub Form_Load()

    Dim spkrs As DgnStrings             'contains speaker names
    Dim i As Integer

    Me.Enabled = False
```

```
                                        'check to see if any speech engines are installed
    If DgnEngineControl1.SpeechEngines.Count = 0 Then

        MsgBox "A speach engine is not currently installed on this machine.  Please install one and
try again.", vbOKOnly, "Error"
```

```
    End If

    Unload Me
    Set spkrs = DgnEngineControl1.Speakers

                                        'enters names of speakers into speaker list box
    For i = 1 To spkrs.Count

        frmSpeaker.lstSpeaker.AddItem spkrs(i)

    Next

    frmSpeaker.Show
    Me.Hide
    frmSpeaker.SetFocus

                                        'select first speaker
    If frmSpeaker.lstSpeaker.ListCount > 0 Then

        frmSpeaker.lstSpeaker.Selected(0) = True

    End If

    frmSpeaker.lstSpeaker.SetFocus

End Sub

'--------------------------------------------------------------------------
'
'Function:  cmdRun_Click
'
'Purpose:   Manually runs a command
'
'--------------------------------------------------------------------------

Private Sub Form_Unload(Cancel As Integer)

    If Socket1.Connected Then               'disconnect socket

        Socket1.Disconnect

    End If

    If activated Then                       'check if form was activated

                                            'check if microphone was on
        If DgnMicBtn1.MicState = dgnmicOn Then

            DgnMicBtn1.MicState = dgnmicOff

        End If



                                            'check if speaker files have been modified



        If DgnEngineControl1.SpeakerModified Then

            If MsgBox("Do you want to save your speech files?", vbYesNo, "Save Speach Files") =
vbYes Then

                DgnEngineControl1.SpeakerSave

            End If

        End If
```

```
    End If

End Sub

'------------------------------------------------------------------------
'
'Function:  mnuAudio_Click
'
'Purpose:   Runs audio setup wizard.
'
'------------------------------------------------------------------------

Private Sub mnuAudio_Click()

    Me.Enabled = False
    DgnEngineControl1.AudioSetupWizard ""   'run audio setup wizard

End Sub

'------------------------------------------------------------------------
'
'Function:  mnuCreate_Click
'
'Purpose:   Calls form to create a new user.
'
'------------------------------------------------------------------------

Private Sub mnuCreate_Click()

    Me.Enabled = False
    frmCreate.Show                          'call from to create new user
    frmCreate.SetReturn (1)

End Sub

'------------------------------------------------------------------------
'
'Function:  mnuEditLists_Click
'
'Purpose:   Calls form to edit data lists
'
'------------------------------------------------------------------------

Private Sub mnuEditLists_Click()

    Dim i As Integer

    For i = 0 To List1.ListCount - 1    'enter data into list on form

        frmEdit.lstBox1.AddItem (List1.List(i))

    Next

    For i = 0 To List2.ListCount - 1    'enter data into list on form

        frmEdit.lstBox2.AddItem (List2.List(i))

    Next

    For i = 0 To List3.ListCount - 1    'enter data into lists on form

        frmEdit.lstBox3.AddItem (List3.List(i))
```

```
    For i = 0 To List4.ListCount - 1    'enter data into lists on form
    Nextfrm Edit.lstBox4.AddItem (List4.List(i))

    Next

    Me.Enabled = False
    frmEdit.Show

End Sub

'------------------------------------------------------------------------
'
'Function:  mnuGenTrain_Click
'
'Purpose:   Runs general training.
'
'------------------------------------------------------------------------

Private Sub mnuGenTrain_Click()

    Me.Enabled = False
    DgnEngineControl1.GeneralTraining ""    'run general training

End Sub

'------------------------------------------------------------------------
'
'Function:  mnuNewConnection_Click()
'
'Purpose:   Connects to a new sequencer
'
'------------------------------------------------------------------------

Private Sub mnuNewConnection_Click()

    frmConnect.Show

End Sub

'------------------------------------------------------------------------
'
'Function:  mnuSave_Click
'
'Purpose:   Manually runs a command
'
'------------------------------------------------------------------------
```

```
Private Sub mnuSave_Click()

    Me.Enabled = False

                                    'display wait dialog
    frmWait.Caption = "Saving Speaker"
    frmWait.lblWait.Caption = "Saving " + frmWait.lblWait.Caption
    frmWait.Show

    DgnEngineControl1.SpeakerSave   'save speaker

    Me.Enabled = True
    Me.SetFocus
    Unload frmWait
```

```
End Sub

'------------------------------------------------------------------------
'
'Function:  mnuSelect_Click
'
'Purpose:   Selects a speaker file to use.
'
'------------------------------------------------------------------------

Private Sub mnuSelect_Click()

    Dim spkrs As DgnStrings         'contains names of speaker
    Dim i As Integer

    Set spkrs = DgnEngineControl1.Speakers

    For i = 1 To spkrs.Count

        frmSelect.lstSpeaker.AddItem spkrs(i)

    Next

    Me.Enabled = False
    frmSelect.Show

End Sub

'------------------------------------------------------------------------
'Purpose:
'
'
'
'------------------------------------------------------------------------

Private Sub smnuExit_Click()
    Unload Me
End Sub

'------------------------------------------------------------------------
'
'Function:  CheckList
'
'Purpose:   Checks the word recieved for a match in one of the lists
'           if one is found, then the word is placed in the appropriate
'           text box
'
'Variables: OneWord - Holds a word to be compared against the lists



'------------------------------------------------------------------------



Public Sub CheckList(OneWord As String)
    Dim i As Integer


    For i = 0 To List1.ListCount - 1        'Go through each item in the list
        If List1.List(i) = OneWord Then     'If the word in the list matches
            txtLoc = OneWord                'the word sent to the procedure
            Exit Sub                        'add it to the text box and exit
        End If
    Next i

    For i = 0 To List2.ListCount - 1        'Go through each item in the list
        If List2.List(i) = OneWord Then     'If the word in the list matches
            txtPar = OneWord                'the word sent to the procedure
```

```
        End If
    Next i

    Exit Sub                            'add it to the text box and exit
    For i = 0 To List3.ListCount - 1    'Go through each item in the list
        If List3.List(i) = OneWord Then 'If the word in the list matches
            txtAtt = OneWord            'the word sent to the procedure
            Exit Sub                    'add it to the text box and exit
        End If
    Next i

    For i = 0 To List4.ListCount - 1    'Go through each item in the list
        If List4.List(i) = OneWord Then 'If the word in the list matches
            txtAct = OneWord            'the word sent to the procedure
            Exit Sub                    'add it to the text box and exit
        End If
    Next i
End Sub
```

Figure 8 - Wait Form Source Code

```
VERSION 5.00

Begin VB.Form frmWait

   ClientHeight    =   1770

   ClientLeft      =   60

   ClientTop       =   345
   ClientWidth     =   5400
   LinkTopic       =   "Form1"
   MaxButton       =   0   'False
   MinButton       =   0   'False
   ScaleHeight     =   1770
   ScaleWidth      =   5400
   StartUpPosition =   1  'CenterOwner
   Begin VB.Label lblWait
      Caption         =   "Speaker     Please Wait..."
      BeginProperty Font
         Name            =   "Times New Roman"
         Size            =   18
         Charset         =   0
         Weight          =   400
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
      Height          =   975
      Left            =   1440
      TabIndex        =   0
      Top             =   360
      Width           =   2775
   End
End
Attribute VB_Name = "frmWait"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
```

Figure 9 - Speaker Form Source Code

```
VERSION 5.00

Begin VB.Form frmSpeaker

   Caption         =   "Speaker Selection "

   ClientHeight    =   2625

   ClientLeft      =   60

   ClientTop       =   345

   ClientWidth     =   4875

   LinkTopic       =   "Form1"
   MaxButton       =   0   'False
   ScaleHeight     =   2625
   ScaleWidth      =   4875
   StartUpPosition =   2  'CenterScreen
   Begin VB.CommandButton cmdSelect
      Caption         =   "Select Speaker"
      Default         =   -1  'True
      Height          =   375
      Left            =   3360
      TabIndex        =   4
      Top             =   480
      Width           =   1335
   End
   Begin VB.CommandButton cmdDelete
      Caption         =   "Delete Speaker"
      Height          =   375
      Left            =   3360
      TabIndex        =   3
      Top             =   1440
      Width           =   1335
   End


   Begin VB.CommandButton cmdCreate



      Caption         =   "Create Speaker"
      Height          =   375
      Left            =   3360
      TabIndex        =   2
      Top             =   960
      Width           =   1335
   End
   Begin VB.CommandButton cmdCancel
      Caption         =   "Cancel"
      Height          =   375
      Left            =   3360
      TabIndex        =   1
      Top             =   1920
      Width           =   1335
   End
   Begin VB.ListBox lstSpeaker
      Height          =   1815
      Left            =   240
      TabIndex        =   0
      Top             =   480
      Width           =   2775
   End
```

```
      Caption         =   "Speaker:"
      Height          =   255
      Left            =   240
   Begin VB.Label lblUser
      TabIndex        =   5
      Top             =   120
      Width           =   735
   End
End
Attribute VB_Name = "frmSpeaker"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Dim complete As Boolean                'is true if can be exited without exiting full program

'-------------------------------------------------------------------------
'
'Function:  cmdCancel_Click
'
'Purpose:   Exits speaker select
'
'-------------------------------------------------------------------------

Private Sub cmdCancel_Click()

    Unload Me

End Sub

'-------------------------------------------------------------------------
'
'Function:  cmdCreate_Click
'
'Purpose:   Calls form to create a new user
'
'-------------------------------------------------------------------------

Private Sub cmdCreate_Click()

    Me.Enabled = False                 'call create new user form


    frmCreate.SetReturn (0)


    frmCreate.Show

End Sub

'-------------------------------------------------------------------------
'
'Function:  cmdDelete_Click
'
'Purpose:   Deletes the selected speaker
'
'-------------------------------------------------------------------------

Private Sub cmdDelete_Click()

    On Error GoTo ErrorHandler

    Dim i As Integer

    If lstSpeaker.SelCount = 0 Then     'check to see if selected speaker

        MsgBox "A speaker was not selected!", vbOKOnly, "Error"
```

```
        i = 0
                                        'get i-value of selected speaker
    Else    While Not (i = lstSpeaker.ListCount) And Not (lstSpeaker.Selected(i))

            i = i + 1

        Wend

                                        'verify that user wants to delete speaker
        If MsgBox("Are you sure you want to delete speaker: " + lstSpeaker.List(i), vbYesNo, "Delete
User") = vbYes Then

                                        'delete speaker
            frmmain.DgnEngineControl1.SpeakerDelete lstSpeaker.List(i)
            lstSpeaker.RemoveItem (i)

        End If

    End If

    GoTo CmdExit

ErrorHandler:

                                        'display error message
    MsgBox Str(Err) + " - " + Error$, vbOKOnly, "Error" ' show message

CmdExit:

End Sub

'---------------------------------------------------------------------------
'
'Function:  activated
'
'Purpose:   Sets complete variable to true
'
'---------------------------------------------------------------------------



Public Sub activated()



    complete = True

End Sub

'---------------------------------------------------------------------------
'
'Function:  cmdSelect_Click
'
'Purpose:   Opens the selected user and runs the main form
'
'---------------------------------------------------------------------------

Private Sub cmdSelect_Click()

    Dim i As Integer

    If lstSpeaker.SelCount = 0 Then     'check if selected speaker

        MsgBox "A speaker was not selected!", vbOKOnly, "Error"

    Else
```

```
        i = 0

                                    'get i-value of selected user
        While Not (i = lstSpeaker.ListCount) And Not (lstSpeaker.Selected(i))

            i = i + 1

        Wend

                                    'display wait dialog
        frmWait.Caption = "Loading Speaker:  " + lstSpeaker.List(i)
        frmWait.lblWait.Caption = "Loading " + frmWait.lblWait.Caption
        frmWait.Show

                                    'load speaker
        frmmain.DgnEngineControl1.Speaker = lstSpeaker.List(i)


                                    'check if speaker has completed an audio setup
'        If Not frmmain.DgnEngineControl1.AudioSetupComplete Then

'            MsgBox "You have not completed the audio setup.  You will be given a chance to do so
now.", vbOKOnly, "Incomplete Audio Setup"

'            frmSpeaker.Enabled = False

'            frmCreate.Show              'run create user to finish audio setup
'            frmCreate.SetName (lstSpeaker.List(i))
'            frmCreate.txtSpeaker.Text = lstSpeaker.List(i)
'            frmCreate.SetReturn (2)
'            frmCreate.OffState1
'            frmCreate.State3
'            frmCreate.SetState (3)

'            Exit Sub

'        End If

                                    'check if speaker has been calibrated
        If Not frmmain.DgnEngineControl1.SpeakerCalibrated Then




            Unload frmWait

            MsgBox "You have not gone through general training yet.  You must do this to calibrate
your speach files.  You will be given the chance to do so now.", vbOKOnly, "Speaker Uncalibrated"

            frmSpeaker.Enabled = False

            frmCreate.Show              'run create user to finish calibration
            frmCreate.SetReturn (2)
            frmCreate.OffState1
            frmCreate.State4
            frmCreate.SetState (4)
            frmCreate.SetName (lstSpeaker.List(i))
            frmCreate.txtSpeaker.Text = lstSpeaker.List(i)

            Exit Sub

        End If

        Unload frmWait

        frmmain.Enabled = True
```

```
        frmmain.Activate
        complete = True
        Unload Me

    End If

End Sub

'--------------------------------------------------------------------------
'
'Function:  Form_Unload
'
'Purpose:   Unloads the form
'
'--------------------------------------------------------------------------

Private Sub Form_Unload(Cancel As Integer)

    If Not complete Then            'check if completed speaker selection

        Unload frmmain

    End If

End Sub

'--------------------------------------------------------------------------
'
'Function:  lstSpeaker_DblClick
'
'Purpose:   Selects the user by double clicking on list
'
'--------------------------------------------------------------------------




Private Sub lstSpeaker_DblClick()

    cmdSelect_Click

End Sub
```

Figure 10 - Select Form Source Code

```
VERSION 5.00

Begin VB.Form frmSelect

   Caption         =   "Select Speaker"

   ClientHeight    =   2925

   ClientLeft      =   60

   ClientTop       =   345
```

```
   LinkTopic       =   "Form1"
   MaxButton       =   0   'False
   ClientWidth     =   4110
   ScaleHeight     =   2925
   ScaleWidth      =   4110
   ShowInTaskbar   =   0   'False
   StartUpPosition =   2  'CenterScreen
   Begin VB.CommandButton cmdCancel
      Caption         =   "&Cancel"
      Height          =   375
      Left            =   2520
      TabIndex        =   2
      Top             =   2280
      Width           =   1215
   End
   Begin VB.CommandButton cmdSelect
      Caption         =   "&Select"
      Default         =   -1  'True
      Height          =   375
      Left            =   2520
      TabIndex        =   1
      Top             =   1800
      Width           =   1215
   End
   Begin VB.ListBox lstSpeaker
      Height          =   2010
      Left            =   360
      TabIndex        =   0
      Top             =   600
      Width           =   1815
   End
   Begin VB.Label lblSpeaker
      Caption         =   "Speaker:"
```

```
      Height          =   255
```

```
      Left            =   360
      TabIndex        =   3
      Top             =   240
      Width           =   855
   End
End
Attribute VB_Name = "frmSelect"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False

'------------------------------------------------------------------------
'
'Function:  cmdCancel_Click
'
'Purpose:   Cancels form
'
'------------------------------------------------------------------------

Private Sub cmdCancel_Click()

    frmmain.Enabled = True          'set focus back to the main form and exit
    frmmain.SetFocus
    Unload Me

End Sub

'------------------------------------------------------------------------
```

```
'Function:  cmdSelect_Click
'
'Purpose:   Selects a speaker and load that speakers speech files
'
'-------------------------------------------------------------------------

Private Sub cmdSelect_Click()

    Dim i As Integer

    If lstSpeaker.SelCount = 0 Then 'check to see if a speaker was selected

        MsgBox "A speaker was not selected!", vbOKOnly, "Error"

    Else

                                    'check to see if user wants to save the current speaker
        If MsgBox("Do you want to save your current speaker?", vbYesNo, "Save Speaker") = vbYes Then

            Me.Enabled = False      'display wait dialog
            frmWait.Caption = "Saving Speaker"
            frmWait.lblWait.Caption = "Saving " + frmWait.lblWait.Caption

            frmmain.DgnEngineControl1.SpeakerSave

            Unload frmWait

            Me.Enabled = True
            Me.SetFocus

        End If

        i = 0


                                    'get i-value of selected speaker


        While Not (i = lstSpeaker.ListCount) And Not (lstSpeaker.Selected(i))

            i = i + 1

        Wend

                                    'load speaker
        frmWait.Caption = "Loading Speaker:  " + lstSpeaker.List(i)
        frmWait.lblWait.Caption = "Loading " + frmWait.lblWait.Caption
        frmWait.Show

        frmmain.DgnEngineControl1.Speaker = lstSpeaker.List(i)

        Unload frmWait

                                    'check if audio setup has been completed
        If Not frmmain.DgnEngineControl1.AudioSetupComplete Then

            MsgBox "You have not completed the audio setup.  You will be given a chance to do so
now.", vbOKOnly, "Incomplete Audio Setup"

            frmSelect.Enabled = False

            frmCreate.Show          'call up create speaker form
            frmCreate.SetStatus (True)
            frmCreate.SetName (lstSpeaker.List(i))
            frmCreate.txtSpeaker.Text = lstSpeaker.List(i)
            frmCreate.SetReturn (3)
            frmCreate.OffState1
```

```
            frmCreate.SetState (3)

            Exit Sub
            frmCreate.State3
        End If

                                    'check if speaker has been calibrated
        If Not frmmain.DgnEngineControl1.SpeakerCalibrated Then

            MsgBox "You have not gone through general training yet.  You must do this to calibrate
your speach files.  You will be given the chance to do so now.", vbOKOnly, "Speaker Uncalibrated"

            frmSelect.Enabled = False

            frmCreate.Show          'call up create speaker form
            frmCreate.SetStatus (True)
            frmCreate.SetReturn (3)
            frmCreate.OffState1
            frmCreate.State4
            frmCreate.SetState (4)
            frmCreate.SetName (lstSpeaker.List(i))
            frmCreate.txtSpeaker.Text = lstSpeaker.List(i)

            Exit Sub

        End If

        frmmain.Enabled = True
        frmmain.SetFocus

        Unload Me

    End If
```

```
End Sub

'---------------------------------------------------------------------------
'
'Function:  lstSpeaker_DblClick
'
'Purpose:   Select speaker when list is double clicked
'
'---------------------------------------------------------------------------

Private Sub lstSpeaker_DblClick()

    cmdSelect_Click

End Sub
```

Figure 11 - Connect Form Source Code

```
VERSION 5.00
```

```
Begin VB.Form frmConnect
   Caption         =   "Connect to Sequencer"
   ClientHeight    =   2610

   ClientLeft      =   60
   ClientTop       =   345
   ClientWidth     =   3885
   LinkTopic       =   "Form1"
   ScaleHeight     =   2610
   ScaleWidth      =   3885
   StartUpPosition =   3  'Windows Default
   Begin VB.TextBox txtPort
      Height          =   375
      Left            =   2040
      TabIndex        =   4
      Top             =   1320
      Width           =   1215
   End
   Begin VB.TextBox txtName
      Height          =   375
      Left            =   2040
      TabIndex        =   3
      Top             =   360
      Width           =   1215
   End
   Begin VB.TextBox txtID
      Height          =   375
      Left            =   2040


      TabIndex        =   2



      Top             =   840
      Width           =   1215
   End
   Begin VB.CommandButton cmdCancel
      Caption         =   "Cancel"
      Height          =   495
      Left            =   2040
      TabIndex        =   1
      Top             =   1920
      Width           =   1215
   End
   Begin VB.CommandButton cmdConnect
      Caption         =   "Connect"
      Height          =   495
      Left            =   720
      TabIndex        =   0
      Top             =   1920
      Width           =   1215
   End
   Begin VB.Line Line3
      X1              =   3360
      X2              =   3360
      Y1              =   480
      Y2              =   1080
   End
   Begin VB.Line Line2
      X1              =   3240
      X2              =   3360
      Y1              =   1080
      Y2              =   1080
   End
   Begin VB.Line Line1
      X1              =   3240
```

```
      Y1              =   480
      Y2              =   480
   End
   Begin VB.Label Label3
      X2              =   3360
      Caption         =   "Port:"
      Height          =   255
      Left            =   720
      TabIndex        =   7
      Top             =   1320
      Width           =   1215
   End
   Begin VB.Label Label2
      Caption         =   "IP Address:"
      Height          =   255
      Left            =   720
      TabIndex        =   6
      Top             =   840
      Width           =   1215
   End
   Begin VB.Label Label1
      Caption         =   "Host Name:"
      Height          =   255
      Left            =   720
      TabIndex        =   5
      Top             =   360
```

```
      Width           =   1215
```

```
   End
End
Attribute VB_Name = "frmConnect"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False

'-----------------------------------------------------------------------
'
'Function:  cmdCancel_Click
'
'Purpose:   Exits form
'
'-----------------------------------------------------------------------

Private Sub cmdCancel_Click()

    frmmain.SetFocus
    Unload Me

End Sub

'-----------------------------------------------------------------------
'
'Function:  cmdConnect_Click
'
'Purpose:   Connects to a new sequencer
'
'-----------------------------------------------------------------------

Private Sub cmdConnect_Click()


    txtName.Text = Trim$(txtName.Text)      'fix variables
    txtID.Text = Trim$(txtID.Text)
    txtPort.TabIndex = Trim$(txtPort.Text)
```

```
                                        'check if have id and port
    If (txtName.Text = "") And (Not (txtID.Text = "")) And (Not (txtPort.Text = "")) Then

        If frmmain.Socket1.Connected Then   'check if already have connection

            frmmain.Socket1.Disconnect

        End If

                                        'set up socket
        frmmain.Socket1.HostAddress = Trim$(txtID.Text)
        frmmain.Socket1.RemotePort = Val(Trim$(txtPort.Text))

                                        'check if connection
        If frmmain.Socket1.Connect <> 0 Then

            MsgBox "Error", "Could not connect to sequencer!"
            Exit Sub

        End If

    Else

        If Not (txtPort.Text = "") Then     'check if have port and host name
```

```
                                        'check if already connected
```

```
            If frmmain.Socket1.Connected Then

                frmmain.Socket1.Disconnect

            End If

                                        'set up socket
            frmmain.Socket1.Hostname = Trim$(txtName.Text)
            frmmain.Socket1.RemotePort = Val(Trim$(txtPort.Text))

                                        'check if connected
            If frmmain.Socket1.Connect <> 0 Then

                MsgBox "Error", "Could not connect to sequencer!"
                Exit Sub

           End If

        Else
            Exit Sub

        End If
    End If

    Unload Me

End Sub

'---------------------------------------------------------------------------
'
'Function:  Form_Load
'
'Purpose:   Initializes screen
'
'---------------------------------------------------------------------------

Private Sub Form_Load()
```

```
    txtID.Text = frmmain.Socket1.HostAddress
    txtPort.Text = Trim$(Str$(frmmain.Socket1.RemotePort))

End Sub

'-----------------------------------------------------------------------
'
'Function:  txtId_Change
'
'Purpose:   Removes host name
'
'-----------------------------------------------------------------------

Private Sub txtID_Change()

    txtName.Text = ""

End Sub

'-----------------------------------------------------------------------
'
'Function:  txtName_Change
'
'Purpose:   Removes id
'



'-----------------------------------------------------------------------



Private Sub txtName_Change()

    txtID.Text = ""

End Sub
```

Figure 12 - Edit Form Source Code

```
VERSION 5.00

Begin VB.Form frmEdit

   Caption         =   "Edit Command Lists"

   ClientHeight    =   5655

   ClientLeft      =   60
   ClientTop       =   345
   ClientWidth     =   9045
   LinkTopic       =   "Form1"
   MaxButton       =   0   'False
   ScaleHeight     =   5655
   ScaleWidth      =   9045
   StartUpPosition =   2  'CenterScreen
   Begin VB.TextBox txt4
      Height          =   375
      Left            =   6600
```

```
      Top             =   3120
      Width           =   1815
   End
   Begin VB.TextBox txt3
      Height          =   375
      Left            =   4560
      TabIndex        =   21
      Top             =   3120
      Width           =   1695
   End
   Begin VB.TextBox txt2
      Height          =   375
      Left            =   2520
      TabIndex        =   20
      Top             =   3120
      Width           =   1815
   End
   Begin VB.TextBox txt1
      Height          =   375
      Left            =   480
      TabIndex        =   19
      Top             =   3120
      Width           =   1815
   End
   Begin VB.CommandButton cmdCancel
      Caption         =   "Cancel"
      Height          =   495
```

```
      Left            =   5880
```

```
      TabIndex        =   14
      Top             =   4800
      Width           =   1215
   End
   Begin VB.CommandButton cmdDone
      Caption         =   "Done"
      Default         =   -1  'True
      Height          =   495
      Left            =   7200
      TabIndex        =   13
      Top             =   4800
      Width           =   1215
   End
   Begin VB.CommandButton cmdSave
      Caption         =   "Save"
      Height          =   495
      Left            =   4560
      TabIndex        =   12
      Top             =   4800
      Width           =   1215
   End
   Begin VB.CommandButton cmdAdd4
      Caption         =   "Add Word"
      Height          =   375
      Left            =   6600
      TabIndex        =   11
      Top             =   3600
      Width           =   1815
   End
   Begin VB.CommandButton cmdRemove4
      Caption         =   "Remove Word"
      Height          =   375
      Left            =   6600
      TabIndex        =   10
      Top             =   4080
```

```
End
Begin VB.CommandButton cmdRemove3
   Caption         =   "Remove Word"
   Height          =   375
   Left            =   4560
   TabIndex        =   9
   Top             =   4080
   Width           =   1695
End
Begin VB.CommandButton cmdAdd3
   Caption         =   "Add Word"
   Height          =   375
   Left            =   4560
   TabIndex        =   8
   Top             =   3600
   Width           =   1695
End
Begin VB.CommandButton cmdAdd2
   Caption         =   "Add Word"
   Height          =   375
   Left            =   2520
   TabIndex        =   7
   Top             =   3600
   Width           =   1815
End
Begin VB.CommandButton cmdRemove2
   Caption         =   "Remove Word"
   Height          =   375
   Left            =   2520
   TabIndex        =   6
   Top             =   4080
   Width           =   1815
End
Begin VB.CommandButton cmdAdd1
   Caption         =   "Add Word"
   Height          =   375
   Left            =   480
   TabIndex        =   5
   Top             =   3600
   Width           =   1815
End
Begin VB.CommandButton cmdRemove1
   Caption         =   "Remove Word"
   Height          =   375
   Left            =   480
   TabIndex        =   4
   Top             =   4080
   Width           =   1815
End
Begin VB.ListBox lstBox4
   Height          =   2205
   Left            =   6600
   TabIndex        =   3
   Top             =   720
   Width           =   1815
End
Begin VB.ListBox lstBox3
   Height          =   2205
   Left            =   4560
   TabIndex        =   2
   Top             =   720
   Width           =   1815
End
```

```
      Height          =   2205
      Left            =   2520
      TabIndex        =   1
      Top             =   720
      Width           =   1815
   End
   Begin VB.ListBox lstBox2
      Height          =   2205
      Left            =   480
      TabIndex        =   0
      Top             =   720
      Width           =   1815
   End
   Begin VB.Label Label4
      Caption         =   "Action"
      Height          =   255
      Left            =   6600
      TabIndex        =   18
      Top             =   360
      Width           =   1215
   End
   Begin VB.Label Label3
      Caption         =   "Attribute"
      Height          =   255
      Left            =   4560
      TabIndex        =   17
      Top             =   360
      Width           =   1215
   End
   Begin VB.Label Label2
      Caption         =   "Part"
      Height          =   255
      Left            =   2520
      TabIndex        =   16
      Top             =   360
      Width           =   1215
   End
   Begin VB.Label Label1
      Caption         =   "Location"
      Height          =   255
      Left            =   480
      TabIndex        =   15
      Top             =   360
      Width           =   1215
   End
End
Attribute VB_Name = "frmEdit"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Dim edited As Boolean                   'hold if data has been edited

'------------------------------------------------------------------------
'
'Function:  cmdAdd1_Click
'
'Purpose:   Adds data to the first list box
'
'------------------------------------------------------------------------

Private Sub cmdAdd1_Click()
```

```
        lstBox1.AddItem (txt1.Text)     'add data to list box
        txt1.Text = ""
    If Not txt1.Text = "" Then          'check to see if data was entered
        edited = True

    End If

End Sub

'------------------------------------------------------------------------
'
'Function:  cmdAdd2_Click
'
'Purpose:   Adds data to the second list box
'
'------------------------------------------------------------------------

Private Sub cmdAdd2_Click()

    If Not txt2.Text = "" Then          'check to see if data was entered

        lstBox2.AddItem (txt2.Text)     'add data to list box



        txt2.Text = ""



        edited = True

    End If

End Sub

'------------------------------------------------------------------------
'
'Function:  cmdAdd3_Click
'
'Purpose:   Adds data to the third list box
'
'------------------------------------------------------------------------

Private Sub cmdAdd3_Click()

    If Not txt3.Text = "" Then          'check to see if data was entered

        lstBox3.AddItem (txt3.Text)     'add data to list box
        txt3.Text = ""

        edited = True

    End If

End Sub

'------------------------------------------------------------------------
'
'Function:  cmdAdd4_Click
'
'Purpose:   Adds data to the forth list box
'
'------------------------------------------------------------------------

Private Sub cmdAdd4_Click()

    If Not txt4.Text = "" Then          'check to see if data was entered
```

```
        lstBox4.AddItem (txt4.Text)     'add data to list box
        txt4.Text = ""

        edited = True

    End If

End Sub

'------------------------------------------------------------------------
'
'Function:  cmdCancel_Click
'
'Purpose:   Cancels from the editing form
'
'------------------------------------------------------------------------

Private Sub cmdCancel_Click()

    frmmain.Enabled = True              'enable main form and unload
    frmmain.SetFocus
    edited = False



    Unload Me



End Sub

'------------------------------------------------------------------------
'
'Function:  cmdDone_Click
'
'Purpose:   Adds data to the first list box
'
'------------------------------------------------------------------------

Private Sub cmdDone_Click()
    Dim i As Integer

    If edited Then                      'check to see if data was edited

                                        'check to see if user wants to save edits
        If MsgBox("Do you want to save the word lists?", vbYesNo, "Save Word Lists") = vbYes Then

            frmmain.Enabled = True

            frmmain.List1.Clear         'clear main form lists
            frmmain.List2.Clear
            frmmain.List3.Clear
            frmmain.List4.Clear

                                        'save contents of list1 to main form
            For i = 0 To lstBox1.ListCount - 1

                frmmain.List1.AddItem (lstBox1.List(i))

            Next

                                        'save contents of list2 to main form
            For i = 0 To lstBox2.ListCount - 1

                frmmain.List2.AddItem (lstBox2.List(i))

            Next
```

```
                                    'save contents of list3 to main form
            For i = 0 To lstBox3.ListCount - 1

                frmmain.List3.AddItem (lstBox3.List(i))

            Next

                                    'save contents of list4 to main form
            For i = 0 To lstBox4.ListCount - 1

            frmmain.List4.AddItem (lstBox4.List(i))

            Next

            frmmain.Enabled = False



            edited = False



'
'
'               SAVE TO FILE STUFF GOES HERE
'
'
            Call Save_Files     'Saves the contents of the lists

        End If

    End If

    frmmain.Enabled = True              'set focus back to main form and unload
    frmmain.SetFocus
    Unload Me

End Sub

'---------------------------------------------------------------------------
'
'Function:  RemoveAll
'
'Purpose:   Removes all data from list boxes
'
'---------------------------------------------------------------------------

Private Sub RemoveAll()

    lstBox1.Clear                       'remove all data
    lstBox2.Clear
    lstBox3.Clear
    lstBox4.Clear

End Sub

'---------------------------------------------------------------------------
'
'Function:  cmdRemove1_Click
'
'Purpose:   Removes selelcted item from list box
'
'---------------------------------------------------------------------------

Private Sub cmdRemove1_Click()

    Dim i As Integer
```

```
        i = 0
    If lstBox1.SelCount > 0 Then        'check to see if selected item
        While Not (i = lstBox1.ListCount) And Not (lstBox1.Selected(i))
            i = i + 1
        Wend
        lstBox1.RemoveItem (i)          'remove selected item
        edited = True
    End If
End Sub
```

```
'---------------------------------------------------------------------------
'
'Function:  cmdRemove2_Click
'
'Purpose:   Removes selelcted item from list box
'
'---------------------------------------------------------------------------

Private Sub cmdRemove2_Click()

    Dim i As Integer

                                        'check if selected item
    If lstBox2.SelCount > 0 Then

        i = 0

                                        'gets i-value of selected item
        While Not (i = lstBox2.ListCount) And Not (lstBox2.Selected(i))

            i = i + 1

        Wend

        lstBox2.RemoveItem (i)          'remove selected item
        edited = True

    End If

End Sub

'---------------------------------------------------------------------------
'
'Function:  cmdRemove3_Click
'
'Purpose:   Removes selelcted item from list box
'
'---------------------------------------------------------------------------

Private Sub cmdRemove3_Click()

    Dim i As Integer

    If lstBox3.SelCount > 0 Then        'check if selected item

        i = 0
```

```
                                        'gets i-value of selected item
        While Not (i = lstBox3.ListCount) And Not (lstBox3.Selected(i))

            i = i + 1

        Wend

        lstBox3.RemoveItem (i)          'remove selected item
        edited = True

    End If

End Sub

'---------------------------------------------------------------------------
'



'Function:  cmdRemove4_Click



'
'Purpose:   Removes selelcted item from list box
'
'---------------------------------------------------------------------------

Private Sub cmdRemove4_Click()

    Dim i As Integer

    If lstBox4.SelCount > 0 Then        'checks if selected item

        i = 0
                                        'gets i-value of selected item
        While Not (i = lstBox4.ListCount) And Not (lstBox4.Selected(i))

            i = i + 1

        Wend

        lstBox4.RemoveItem (i)          'remove selected item
        edited = True

    End If

End Sub

'---------------------------------------------------------------------------
'
'Function:  cmdSave_Click
'
'Purpose:   Saves data.
'
'---------------------------------------------------------------------------

Private Sub cmdSave_Click()

    Dim i As Integer

    frmmain.Enabled = True

    frmmain.List1.Clear                 'clears all data in main form
    frmmain.List2.Clear
    frmmain.List3.Clear
    frmmain.List4.Clear

    For i = 0 To lstBox1.ListCount - 1  'save data to main form
```

```
            frmmain.List1.AddItem (lstBox1.List(i))
        Next
        For i = 0 To lstBox2.ListCount - 1  'save data to main form
            frmmain.List2.AddItem (lstBox2.List(i))
        Next
        For i = 0 To lstBox3.ListCount - 1  'save data to main form
            frmmain.List3.AddItem (lstBox3.List(i))
        Next


        For i = 0 To lstBox4.ListCount - 1  'save data to main form


            frmmain.List4.AddItem (lstBox4.List(i))
        Next
        frmmain.Enabled = False
        edited = False
    '
    '
    '               SAVE TO FILE STUFF GOES HERE
    '
    '
        Call Save_Files     'Saves the contents of the lists
    End Sub

    '---------------------------------------------------------------------------
    '
    'Function:  Form_Load
    '
    'Purpose:   Initialize all variables
    '
    '---------------------------------------------------------------------------

    Private Sub Form_Load()

        edited = False

    End Sub

    '---------------------------------------------------------------------------
    '
    'Function:  Form_Unload
    '
    'Purpose:   Removes all data from list boxes
    '
    '---------------------------------------------------------------------------

    Private Sub Form_Unload(Cancel As Integer)

        RemoveAll

    End Sub

    '---------------------------------------------------------------------------
    '
```

```
'
'Purpose:   Writes each list to a file so that it is saved for
'           later access
'Function:  Save_Files
'-----------------------------------------------------------

Private Sub Save_Files()
    Dim fnum As Integer
    Dim i As Integer

        fnum = FreeFile



        Open App.Path & "\List1.dat" For Output As fnum



                        'sets the variable for output to a file
        For i = 0 To frmmain.List1.ListCount - 1
            Print #fnum, frmmain.List1.List(i)
        Next i          'goes through the list, putting each item
                        'in the file

        Close fnum      'close the file after output

        fnum = FreeFile
        Open App.Path & "\List2.dat" For Output As fnum
                        'sets the variable for output to a file
        For i = 0 To frmmain.List2.ListCount - 1
            Print #fnum, frmmain.List2.List(i)
        Next i          'goes through the list, putting each item
                        'in the file

        Close fnum      'close the file after output

        fnum = FreeFile
        Open App.Path & "\List3.dat" For Output As fnum
                        'sets the variable for output to a file
        For i = 0 To frmmain.List3.ListCount - 1
            Print #fnum, frmmain.List3.List(i)
        Next i          'goes through the list, putting each item
                        'in the file

        Close fnum      'close the file after output

        fnum = FreeFile
        Open App.Path & "\List4.dat" For Output As fnum
                        'sets the variable for output to a file
        For i = 0 To frmmain.List4.ListCount - 1
            Print #fnum, frmmain.List4.List(i)
        Next i          'goes through the list, putting each item
                        'in the file

        Close fnum      'close the file after output

End Sub
```

Figure 13 - Create Form Source Code

```
VERSION 5.00
```

```
Begin VB.Form frmCreate
   BorderStyle     =   1  'Fixed Single
   Caption         =   "Create A New User"
Object = "{5C486340-2F92-11D1-A47C-00A024A3A678}#1.0#0"; "DNSTK10.DLL"
   ClientHeight    =   3750
   ClientLeft      =   45
   ClientTop       =   330
   ClientWidth     =   7230


   KeyPreview      =   -1  'True


   LinkTopic       =   "Form1"
   MaxButton       =   0   'False
   ScaleHeight     =   3750
   ScaleWidth      =   7230
   ShowInTaskbar   =   0   'False
   StartUpPosition =   2  'CenterScreen
   Begin DNSToolsCtl.DgnEngineControl DgnEngineControl1
      Left            =   6000
      OleObjectBlob   =   "create.frx":0000
      Top             =   2880
   End
   Begin VB.CommandButton cmdFinish
      Caption         =   "Finish Setup"
      Enabled         =   0   'False
      Height          =   375
      Left            =   2880
      TabIndex        =   13
      Top             =   1680
      Visible         =   0   'False
      Width           =   2775
   End
   Begin VB.CommandButton cmdGenTrain
      Caption         =   "Run General Training"
      Height          =   375
      Left            =   2880
      TabIndex        =   11
      Top             =   1080
      Visible         =   0   'False
      Width           =   2775
   End
   Begin VB.CommandButton cmdAudio
      Caption         =   "Run Audio Setup"
      Height          =   375
      Left            =   2880
      TabIndex        =   10
      Top             =   1320
      Visible         =   0   'False
      Width           =   2775
   End
   Begin VB.TextBox txtSpeaker
      Height          =   375
      Left            =   3240
      TabIndex        =   8
      Top             =   1320
      Visible         =   0   'False
      Width           =   3255
   End
   Begin VB.Frame frmMenu
      Height          =   2775
      Left            =   120
      TabIndex        =   3
      Top             =   0
      Width           =   1575
      Begin VB.Label lblMenu4
         Caption         =   "Train Voice Recognition Engine"
```

```
      Left            =   120
      TabIndex        =   7
      Top             =   1920
      Height          =   735
      Width           =   1215
   End
   Begin VB.Label lblMenu3
      Caption         =   "Run Audio Setup Wizard"


      Height          =   495


      Left            =   120
      TabIndex        =   6
      Top             =   1320
      Width           =   1215
   End
   Begin VB.Label lblMenu2
      Caption         =   "Create User Speech File"
      Height          =   495
      Left            =   120
      TabIndex        =   5
      Top             =   720
      Width           =   1215
   End
   Begin VB.Label lblMenu1
      Caption         =   "Welcome"
      BeginProperty Font
         Name            =   "MS Sans Serif"
         Size            =   8.25
         Charset         =   0
         Weight          =   700
         Underline       =   0   'False
         Italic          =   0   'False
         Strikethrough   =   0   'False
      EndProperty
      Height          =   255
      Left            =   120
      TabIndex        =   4
      Top             =   360
      Width           =   1215
   End
End
Begin VB.CommandButton cmdCancel
   Caption         =   "Cancel"
   Height          =   375
   Left            =   4680
   TabIndex        =   2
   Top             =   3240
   Width           =   1215
End
Begin VB.CommandButton cmdNext
   Caption         =   "Next&>"
   Height          =   375
   Left            =   2400
   TabIndex        =   1
   Top             =   3240
   Width           =   1215
End
Begin VB.CommandButton cmdBack
   Caption         =   "&<Back"
   Enabled         =   0   'False
   Height          =   375
   Left            =   1200
   TabIndex        =   0
   Top             =   3240
   Width           =   1215
```

```
      Begin VB.Label lblWelcome
         Caption         =   $"create.frx":0024
         Height          =   2175
      End
         Left            =   2160
         TabIndex        =   12
         Top             =   480
         Width           =   4575



      End



      Begin VB.Label lblSpeaker
         Caption         =   "User Name:"
         Height          =   255
         Left            =   2040
         TabIndex        =   9
         Top             =   1440
         Visible         =   0   'False
         Width           =   975
      End
   End
   Attribute VB_Name = "frmCreate"
   Attribute VB_GlobalNameSpace = False
   Attribute VB_Creatable = False
   Attribute VB_PredeclaredId = True
   Attribute VB_Exposed = False
   Dim state As Integer                    'current create screen
   Dim speaker_name As String              'created speaker name
   Dim return_form As Integer              'name of form to return to when finished
   Dim opened As Boolean                   'if the main form is already opened
   Dim valid As Boolean                    'true if audio setup has been completed
   Dim created As Boolean                  'a user has been created

   Const retSpeaker = 0                    'return form constants
   Const retMain = 1
   Const retSpeaker2 = 2
   Const retCreate = 3

   '-------------------------------------------------------------------------
   '
   'Function:  cmdAudio_Click

   'Purpose:   runs audio setup wizard when audio button is clicked
   '
   '-------------------------------------------------------------------------

   Private Sub cmdAudio_Click()

       DgnEngineControl1.AudioSetupWizard ""

   End Sub

   '-------------------------------------------------------------------------
   '
   'Function:  SetReturn
   '
   'Purpose:   Sets the return form variable
   '
   'Variables: return_to - number to set return form variable to
   '
   '-------------------------------------------------------------------------

   Public Sub SetReturn(return_to As Integer)

       return_form = return_to
```

```
'------------------------------------------------------------------------
'
End Sub
'Function:  cmdBack_Click
'
'Purpose:   Changes the screen state to the previous screen when the
'           back button is pressed.


'


'------------------------------------------------------------------------

Private Sub cmdBack_Click()

    If state = 2 Then               'set second screen up

        OffState2
        State1

    Else

        If state = 3 Then           'set thirds screen up

            OffState3
            State2

        Else

            If state = 4 Then       'set fourth screen up

                OffState4
                State3

            End If

        End If

    End If

    state = state - 1

End Sub

'------------------------------------------------------------------------
'
'Function:  SetState
'
'Purpose:   Sets the number of the current screen state.
'
'Variables: s - the number to set the current state to
'
'------------------------------------------------------------------------

Public Sub SetState(s As Integer)

    state = s

End Sub

'------------------------------------------------------------------------
'
'Function:  cmdCancel_Click
'
'Purpose:   Cancels the create speaker process.
'
```

```
'------------------------------------------------------------------------


Private Sub cmdCancel_Click()

                            'check to see if a speaker has been created or is being modified
    If created = True Or return_form = retSpeaker2 Or return_form = retCreate Then

                            'check to see if user wants to cancel
        If MsgBox("If you cancel now your speaker settings will not be saved.  Do you still want to
cancel?", vbYesNo, "Cancel Operation") = vbYes Then

                            'check if called from select speaker function from the main form
            If return_form = retCreate Then

                frmSelect.Enabled = True
                CloseForm

            End If

            frmSpeaker.Enabled = True
            CloseForm       'exit create speaker form

        End If

    Else

    frmSpeaker.Enabled = True
    CloseForm               'exit create speaker form

    End If

End Sub

'------------------------------------------------------------------------
'
'Function:  CloseForm
'
'Purpose:   Sets focus back to called form and exits
'
'------------------------------------------------------------------------

Private Sub CloseForm()

                                        'checks to see if return for is speaker select form
    If return_form = retSpeaker Or return_form = retSpeaker2 Then

        frmSpeaker.SetFocus             'set focus to speaker select form and exit
        frmSpeaker.Enabled = True
        Unload Me

    Else

        If return_form = retCreate Then

            frmSelect.SetFocus          'set focus to select form and exit
            frmSelect.Enabled = True
            Unload Me

        Else

            frmmain.Enabled = True      'set focus to main form and exit
            frmmain.SetFocus
            Unload Me
```

```
        End If


    End If

End Sub

'------------------------------------------------------------------------
'
'Function:  cmdFinish_Click
'
'Purpose:   Finishes creating a speaker, saves the speaker and activates
'           the main form.
'
'------------------------------------------------------------------------

Private Sub cmdFinish_Click()

                                    'displays waiting form
    frmWait.Caption = "Saving Speaker"
    frmWait.lblWait.Caption = "Saving " + frmWait.lblWait.Caption
    Me.Enabled = False
    frmWait.Show

                                    'checks to see if just modifying a speaker
    If return_form = retSpeaker2 Or return_from = retCreate Then

        frmmain.DgnEngineControl1.SpeakerSave

    Else

        DgnEngineControl1.SpeakerSave

    End If

    Unload frmWait                  'close waiting form

    If opened Then                  'check to see if main form already opened

        frmmain.Enabled = True      'sets focus back to main form
        frmmain.SetFocus

    Else

        frmSpeaker.activated        'activates main form and set the focus to it
        Unload frmSpeaker
        frmmain.Activate

    End If

    If return_form = retCreate Then 'check to see if called from select speaker from main form

        Unload frmSelect

    End If

    Unload Me                       'close the form

End Sub

'------------------------------------------------------------------------
'
'Function:  cmdGenTrain_Click
'
'Purpose:   Runs general training
```

```
'
'------------------------------------------------------------------------

Private Sub cmdGenTrain_Click()

    DgnEngineControl1.GeneralTraining ""

End Sub

'------------------------------------------------------------------------
'
'Function:  State1
'
'Purpose:   Sets the screen as the welcome screen
'
'------------------------------------------------------------------------

Private Sub State1()

    cmdBack.Enabled = False
    lblMenu1.FontBold = True
    lblWelcome.Visible = True

End Sub

'------------------------------------------------------------------------
'
'Function:  OffState1
'
'Purpose:   Removes all elements of the first state
'
'
'
'------------------------------------------------------------------------

Public Sub OffState1()

    cmdBack.Enabled = True
    lblMenu1.FontBold = False
    lblWelcome.Visible = False
    cmdNext.Default = True

End Sub

'------------------------------------------------------------------------
'
'Function:  State2
'
'Purpose:   Sets up the screen as the enter speaker name screen.
'
'------------------------------------------------------------------------

Private Sub State2()

    lblSpeaker.Visible = True           'displays elements of set name screen
    txtSpeaker.Visible = True
    lblMenu2.FontBold = True

    If txtSpeaker.Text = "" Then        'check to see if a name has been entered

        cmdNext.Enabled = False

    End If
```

```
    txtSpeaker.SetFocus


End Sub

'------------------------------------------------------------------------
'
'Function:  OffState2
'
'Purpose:   Removes all elements of the second state
'
'------------------------------------------------------------------------

Private Sub OffState2()

    lblSpeaker.Visible = False          'removes all elements from display
    txtSpeaker.Visible = False
    lblMenu2.FontBold = False
    cmdNext.Enabled = True

End Sub

'------------------------------------------------------------------------
'
'Function:  State3
'
'Purpose:   Sets up the screen to display the audio setup screen
'
'------------------------------------------------------------------------

Public Sub State3()

    lblMenu3.FontBold = True            'display audio screen elements
    cmdAudio.Visible = True
    cmdAudio.SetFocus
    cmdAudio.Default = True

                                        'check to see if audio setup has already been run
    If Not DgnEngineControl1.AudioSetupComplete And Not valid Then

        cmdNext.Enabled = False

    End If

End Sub

'------------------------------------------------------------------------
'
'Function:  OffState3
'
'Purpose:   Remove all elements of state 3 from being displayed
'
'------------------------------------------------------------------------

Private Sub OffState3()

    lblMenu3.FontBold = False           'remove elelments
    cmdAudio.Visible = False
    cmdAudio.Default = False
    cmdNext.Enabled = True
```

```
End Sub

'------------------------------------------------------------------------
'
'Function:  State4
'
'Purpose:   Setup general training and final display state
'
'------------------------------------------------------------------------

Public Sub State4()

    cmdNext.Enabled = False             'add elements to display
    lblMenu4.FontBold = True
    cmdGenTrain.Visible = True
    cmdFinish.Visible = True
    cmdGenTrain.Default = True

                                        'check to see if speaker has been calibrated
    If DgnEngineControl1.SpeakerCalibrated Then

        cmdFinish.Enabled = True

    Else

        cmdFinish.Enabled = False

    End If

End Sub

'------------------------------------------------------------------------
'
'Function:  OffState4
'
'Purpose:   Remove all elements of state 4
'
'------------------------------------------------------------------------

Private Sub OffState4()

    cmdNext.Enabled = True                  'remove elements
    lblMenu4.FontBold = False
    cmdGenTrain.Visible = False
    cmdFinish.Visible = False
    cmdFinish.Enabled = False

End Sub

'------------------------------------------------------------------------
'
'Function:  cmdNext_Click
'
'Purpose:   Increments to the next display state when the next button
'           is clicked.
'
'
'------------------------------------------------------------------------

Private Sub cmdNext_Click()
```

```
    On Error GoTo ErrorHandler


    If state = 1 Then               'check if current state is state 1

        OffState1                   'setup state 2
        State2

    Else

        If state = 2 Then           'check if current state is state 2

                                    'check is speaker has already been created
            If Not txtSpeaker.Text = speaker_name Then

                Me.Enabled = False
                created = True

                                    'display waiting dialog
                frmWait.Caption = "Creating User: " + txtSpeaker.Text
                frmWait.lblWait.Caption = "Creating " + frmWait.lblWait.Caption
                frmWait.Show

                                    'create speaker
                DgnEngineControl1.SpeakerCreate txtSpeaker.Text
                speaker_name = txtSpeaker.Text
                DgnEngineControl1.Speaker = txtSpeaker.Text

                Unload frmWait

                Me.Enabled = True
                Me.SetFocus

                                    'check to see if called from select speaker form
                If return_form = retSpeaker Then

                    frmSpeaker.lstSpeaker.AddItem (txtSpeaker.Text)

                End If

            End If

            OffState2               'setup state 3
            State3

        Else

            If state = 3 Then       'check for current state as state 3

                OffState3           'setup state 4
                State4

            End If

        End If

    End If

    state = state + 1

    GoTo ExitSub

ErrorHandler:                       'handles any error that may happen during run time
```

```
                                    'check to see error is "user already exists"
    If Err.Number = -2147220444 Then

        Unload frmWait              'display error dialog
        MsgBox "User already exists.", vbOKOnly, "Error"
        Me.Enabled = True
        Me.SetFocus

        Exit Sub

    Else
                                    'display general error message and exit
        MsgBox Str(Err) + " - " + Error$, vbOKOnly, "Error" ' show message

        Unload Me

    End If

ExitSub:

End Sub

'---------------------------------------------------------------------------
'
'Function:  DgnEnginecontrol1_DialogClosed
'
'Purpose:   Is called when a DgnEngineControl function exits.
'
'Variables: Dialog   - specifies which dialog has exited
'           ExitCode - hold exit status of dialog
'
'---------------------------------------------------------------------------

Private Sub DgnEngineControl1_DialogClosed(Dialog As DNSToolsCtl.DgnDialogConstants, ExitCode As
Long)

                                        'check if audio wizard has closed
    If Dialog = dgndlgAudioSetupWizard Then

                                        'check if audio setup has been completed
        If DgnEngineControl1.AudioSetupComplete Then

            cmdNext.Enabled = True      'enable next button
            cmdNext.Default = True

        Else

                                        'check if setup has been completed but has not yet been set
            If ExitCode = 0 Then

                valid = True            'enable next button
                cmdNext.Enabled = True
                cmdNext.Default = True

            End If

        End If

    Else

                                        'check if general training has closed
        If Dialog = dgndlgGeneralTraining Then
```

```
                                    'check if speaker has been calibrated
            If DgnEngineControl1.SpeakerCalibrated Then

                cmdFinish.Enabled = True 'enable finish button

            End If

        End If

    End If

End Sub

'-------------------------------------------------------------------------
'
'Function:  SetName
'
'Purpose:   Saves the name of the user being created.
'
'Variables: name - contains the name to save
'
'-------------------------------------------------------------------------

Public Sub SetName(name As String)

    speaker_name = name

End Sub

'-------------------------------------------------------------------------
'
'Function:  Save
'
'Purpose:   Saves the current speaker's speach files
'
'-------------------------------------------------------------------------

Private Sub Save()

    DgnEngineControl1.SpeakerSave

End Sub

'-------------------------------------------------------------------------
'
'Function:  Form_Load
'
'Purpose:   Initailizes all setting for creating a speaker
'
'-------------------------------------------------------------------------

Private Sub Form_Load()

    state = 1                           'initialize variables
    speaker_name = ""
    opened = False
    valid = False
    created = False

End Sub

'-------------------------------------------------------------------------
'
```

```
'Function:  SetStatus
'
'Purpose:   Sets the current status of the form
'
'---------------------------------------------------------------------------

Public Sub SetStatus(status As Boolean)

    opened = status

End Sub

'---------------------------------------------------------------------------
'
'Function:  txtSpeaker_Change
'
'Purpose:   Checks to see if the a speaker name has been entered into the
'           speaker name field
'
'
'
'---------------------------------------------------------------------------

Private Sub txtSpeaker_Change()

                                    'check if text box is empty
    If Not txtSpeaker.Text = "" Then

        cmdNext.Enabled = True      'enable next button
        cmdNext.Default = True

    Else

        cmdNext.Enabled = False     'disable next button

    End If

End Sub
```

Figure 14 - Parser and TCP Module Code

```
Attribute VB_Name = "mdlParser"

'--------------------------------------------------------------------------------

'

' Catalyst SocketWrench 2.15
' Copyright 1995-1998, Catalyst Development Corp. All rights reserved.
'
' This file contains the constants and function declarations used
' with the SocketWrench control for Visual Basic 5.0
```

```
'
'-------------------------------------------------------------------------------

'
' General constants used with most of the controls
'
Public Const INVALID_HANDLE = -1
Public Const CONTROL_ERRIGNORE = 0
Public Const CONTROL_ERRDISPLAY = 1

'
' SocketWrench Control Actions
'
Public Const SOCKET_OPEN = 1
Public Const SOCKET_CONNECT = 2
Public Const SOCKET_LISTEN = 3
Public Const SOCKET_ACCEPT = 4
Public Const SOCKET_CANCEL = 5
Public Const SOCKET_FLUSH = 6
Public Const SOCKET_CLOSE = 7
Public Const SOCKET_DISCONNECT = 7
Public Const SOCKET_ABORT = 8
'
' SocketWrench Control States
'
Public Const SOCKET_NONE = 0
Public Const SOCKET_IDLE = 1
Public Const SOCKET_LISTENING = 2
Public Const SOCKET_CONNECTING = 3
Public Const SOCKET_ACCEPTING = 4
Public Const SOCKET_RECEIVING = 5
Public Const SOCKET_SENDING = 6
Public Const SOCKET_CLOSING = 7
'
' Socket Address Families
'
Public Const AF_UNSPEC = 0
Public Const AF_UNIX = 1
Public Const AF_INET = 2
'
' Socket Types
'
Public Const SOCK_STREAM = 1
Public Const SOCK_DGRAM = 2
Public Const SOCK_RAW = 3
Public Const SOCK_RDM = 4
Public Const SOCK_SEQPACKET = 5
'
' Protocol Types
'
Public Const IPPROTO_IP = 0
Public Const IPPROTO_ICMP = 1
Public Const IPPROTO_GGP = 2
Public Const IPPROTO_TCP = 6
Public Const IPPROTO_PUP = 12
Public Const IPPROTO_UDP = 17
Public Const IPPROTO_IDP = 22
Public Const IPPROTO_ND = 77
Public Const IPPROTO_RAW = 255
Public Const IPPROTO_MAX = 256
'
' Well-Known Port Numbers
'
Public Const IPPORT_ANY = 0
```

```
Public Const IPPORT_ECHO = 7
Public Const IPPORT_DISCARD = 9
Public Const IPPORT_SYSTAT = 11
Public Const IPPORT_DAYTIME = 13
Public Const IPPORT_NETSTAT = 15
Public Const IPPORT_FTP = 21
Public Const IPPORT_TELNET = 23
Public Const IPPORT_SMTP = 25
Public Const IPPORT_TIMESERVER = 37
Public Const IPPORT_NAMESERVER = 42
Public Const IPPORT_WHOIS = 43
Public Const IPPORT_MTP = 57
Public Const IPPORT_FINGER = 79
Public Const IPPORT_HTTP = 80
Public Const IPPORT_TFTP = 69
Public Const IPPORT_RESERVED = 1024
Public Const IPPORT_USERRESERVED = 5000
'
' Network Addresses
'
Public Const INADDR_ANY = "0.0.0.0"
Public Const INADDR_LOOPBACK = "127.0.0.1"
Public Const INADDR_NONE = "255.255.255.255"
'
' Shutdown Values
'
Public Const SOCKET_READ = 0
Public Const SOCKET_WRITE = 1
Public Const SOCKET_READWRITE = 2
'
' SocketWrench Error Response
'
Public Const SOCKET_ERRIGNORE = 0
Public Const SOCKET_ERRDISPLAY = 1
'
' SocketWrench Error Codes
'
Public Const WSABASEERR = 24000
Public Const WSAEINTR = 24004
Public Const WSAEBADF = 24009
Public Const WSAEACCES = 24013
Public Const WSAEFAULT = 24014
Public Const WSAEINVAL = 24022
Public Const WSAEMFILE = 24024
Public Const WSAEWOULDBLOCK = 24035
Public Const WSAEINPROGRESS = 24036
Public Const WSAEALREADY = 24037
Public Const WSAENOTSOCK = 24038
Public Const WSAEDESTADDRREQ = 24039
Public Const WSAEMSGSIZE = 24040
Public Const WSAEPROTOTYPE = 24041
Public Const WSAENOPROTOOPT = 24042
Public Const WSAEPROTONOSUPPORT = 24043
Public Const WSAESOCKTNOSUPPORT = 24044
Public Const WSAEOPNOTSUPP = 24045
Public Const WSAEPFNOSUPPORT = 24046
Public Const WSAEAFNOSUPPORT = 24047
Public Const WSAEADDRINUSE = 24048
Public Const WSAEADDRNOTAVAIL = 24049
Public Const WSAENETDOWN = 24050
Public Const WSAENETUNREACH = 24051
Public Const WSAENETRESET = 24052
Public Const WSAECONNABORTED = 24053
Public Const WSAECONNRESET = 24054
```

```
Public Const WSAENOBUFS = 24055
Public Const WSAEISCONN = 24056
Public Const WSAENOTCONN = 24057
Public Const WSAESHUTDOWN = 24058
Public Const WSAETOOMANYREFS = 24059
Public Const WSAETIMEDOUT = 24060
Public Const WSAECONNREFUSED = 24061
Public Const WSAELOOP = 24062
Public Const WSAENAMETOOLONG = 24063
Public Const WSAEHOSTDOWN = 24064
Public Const WSAEHOSTUNREACH = 24065
Public Const WSAENOTEMPTY = 24066
Public Const WSAEPROCLIM = 24067
Public Const WSAEUSERS = 24068
Public Const WSAEDQUOT = 24069
Public Const WSAESTALE = 24070
Public Const WSAEREMOTE = 24071
Public Const WSASYSNOTREADY = 24091
Public Const WSAVERNOTSUPPORTED = 24092
Public Const WSANOTINITIALISED = 24093
Public Const WSAHOST_NOT_FOUND = 25001
Public Const WSATRY_AGAIN = 25002
Public Const WSANO_RECOVERY = 25003
Public Const WSANO_DATA = 25004
Public Const WSANO_ADDRESS = 25004

'
' RAS Control Actions
'
Public Const RAS_ACTION_CONNECT = 1
Public Const RAS_ACTION_DISCONNECT = 2
Public Const RAS_ACTION_RESET = 3
'
' RAS Control States
'
Public Const RAS_UNUSED = -1
Public Const RAS_OPENPORT = 0
Public Const RAS_PORTOPENED = 1
Public Const RAS_CONNECTDEV = 2
Public Const RAS_DEVCONNECTED = 3
Public Const RAS_ALLDEVCONNECTED = 4
Public Const RAS_AUTHENTICATE = 5
Public Const RAS_AUTHENTICATED = 14
Public Const RAS_PREPCALLBACK = 15
Public Const RAS_MODEMRESET = 16
Public Const RAS_WAITFORCALL = 17
Public Const RAS_PROJECTED = 18
Public Const RAS_PAUSED = 4096
Public Const RAS_RETRYAUTH = 4097
Public Const RAS_CALLBACK = 4098
Public Const RAS_PASSEXPIRED = 4099
Public Const RAS_CONNECTED = 8192
Public Const RAS_DISCONNECTED = 8193

'
' RAS Control Error Codes
'
' These error codes are returned by the LastError property and
' passed as an argument to the LastError event. These are the
' same codes returned by the RAS library, with 25000 added to the
' base value
'
Public Const ERROR_INVALID_PORT_HANDLE = 25601
Public Const ERROR_PORT_ALREADY_OPEN = 25602
```

```
Public Const ERROR_BUFFER_TOO_SMALL = 25603
Public Const ERROR_WRONG_INFO_SPECIFIED = 25604
Public Const ERROR_CANNOT_SET_PORT_INFO = 25605
Public Const ERROR_PORT_NOT_CONNECTED = 25606
Public Const ERROR_EVENT_INVALID = 25607
Public Const ERROR_DEVICE_DOES_NOT_EXIST = 25608
Public Const ERROR_DEVICETYPE_DOES_NOT_EXIST = 25609
Public Const ERROR_INVALID_BUFFER = 25610
Public Const ERROR_ROUTE_NOT_AVAILABLE = 25611
Public Const ERROR_ROUTE_NOT_ALLOCATED = 25612
Public Const ERROR_INVALID_COMPRESSION_SPECIFIED = 25613
Public Const ERROR_OUT_OF_BUFFERS = 25614
Public Const ERROR_PORT_NOT_FOUND = 25615
Public Const ERROR_ASYNC_REQUEST_PENDING = 25616
Public Const ERROR_ALREADY_DISCONNECTING = 25617
Public Const ERROR_PORT_NOT_OPEN = 25618
Public Const ERROR_PORT_DISCONNECTED = 25619
Public Const ERROR_NO_ENDPOINTS = 25620
Public Const ERROR_CANNOT_OPEN_PHONEBOOK = 25621
Public Const ERROR_CANNOT_LOAD_PHONEBOOK = 25622
Public Const ERROR_CANNOT_FIND_PHONEBOOK_ENTRY = 25623
Public Const ERROR_CANNOT_WRITE_PHONEBOOK = 25624
Public Const ERROR_CORRUPT_PHONEBOOK = 25625
Public Const ERROR_CANNOT_LOAD_STRING = 25626
Public Const ERROR_KEY_NOT_FOUND = 25627
Public Const ERROR_DISCONNECTION = 25628
Public Const ERROR_REMOTE_DISCONNECTION = 25629
Public Const ERROR_HARDWARE_FAILURE = 25630
Public Const ERROR_USER_DISCONNECTION = 25631
Public Const ERROR_INVALID_SIZE = 25632
Public Const ERROR_PORT_NOT_AVAILABLE = 25633
Public Const ERROR_CANNOT_PROJECT_CLIENT = 25634
Public Const ERROR_UNKNOWN = 25635
Public Const ERROR_WRONG_DEVICE_ATTACHED = 25636
Public Const ERROR_BAD_STRING = 25637
Public Const ERROR_REQUEST_TIMEOUT = 25638
Public Const ERROR_CANNOT_GET_LANA = 25639
Public Const ERROR_NETBIOS_ERROR = 25640
Public Const ERROR_SERVER_OUT_OF_RESOURCES = 25641
Public Const ERROR_NAME_EXISTS_ON_NET = 25642
Public Const ERROR_SERVER_GENERAL_NET_FAILURE = 25643
Public Const ERROR_AUTH_INTERNAL = 25645
Public Const ERROR_RESTRICTED_LOGON_HOURS = 25646
Public Const ERROR_ACCT_DISABLED = 25647
Public Const ERROR_PASSWD_EXPIRED = 25648
Public Const ERROR_NO_DIALIN_PERMISSION = 25649
Public Const ERROR_SERVER_NOT_RESPONDING = 25650
Public Const ERROR_FROM_DEVICE = 25651
Public Const ERROR_UNRECOGNIZED_RESPONSE = 25652
Public Const ERROR_MACRO_NOT_FOUND = 25653
Public Const ERROR_MACRO_NOT_DEFINED = 25654
Public Const ERROR_MESSAGE_MACRO_NOT_FOUND = 25655
Public Const ERROR_DEFAULTOFF_MACRO_NOT_FOUND = 25656
Public Const ERROR_FILE_COULD_NOT_BE_OPENED = 25657
Public Const ERROR_DEVICENAME_TOO_LONG = 25658
Public Const ERROR_DEVICENAME_NOT_FOUND = 25659
Public Const ERROR_NO_RESPONSES = 25660


Public Const ERROR_NO_COMMAND_FOUND = 25661


Public Const ERROR_WRONG_KEY_SPECIFIED = 25662
Public Const ERROR_UNKNOWN_DEVICE_TYPE = 25663
```

```
Public Const ERROR_PORT_NOT_CONFIGURED = 25665
Public Const ERROR_DEVICE_NOT_READY = 25666
Public Const ERROR_READING_INI_FILE = 25667
Public Const ERROR_ALLOCATING_MEMORY = 25664
Public Const ERROR_NO_CONNECTION = 25668
Public Const ERROR_BAD_USAGE_IN_INI_FILE = 25669
Public Const ERROR_READING_SECTIONNAME = 25670
Public Const ERROR_READING_DEVICETYPE = 25671
Public Const ERROR_READING_DEVICENAME = 25672
Public Const ERROR_READING_USAGE = 25673
Public Const ERROR_READING_MAXCONNECTBPS = 25674
Public Const ERROR_READING_MAXCARRIERBPS = 25675
Public Const ERROR_LINE_BUSY = 25676
Public Const ERROR_VOICE_ANSWER = 25677
Public Const ERROR_NO_ANSWER = 25678
Public Const ERROR_NO_CARRIER = 25679
Public Const ERROR_NO_DIALTONE = 25680
Public Const ERROR_IN_COMMAND = 25681
Public Const ERROR_WRITING_SECTIONNAME = 25682
Public Const ERROR_WRITING_DEVICETYPE = 25683
Public Const ERROR_WRITING_DEVICENAME = 25684
Public Const ERROR_WRITING_MAXCONNECTBPS = 25685
Public Const ERROR_WRITING_MAXCARRIERBPS = 25686
Public Const ERROR_WRITING_USAGE = 25687
Public Const ERROR_WRITING_DEFAULTOFF = 25688
Public Const ERROR_READING_DEFAULTOFF = 25689
Public Const ERROR_EMPTY_INI_FILE = 25690
Public Const ERROR_AUTHENTICATION_FAILURE = 25691
Public Const ERROR_PORT_OR_DEVICE = 25692
Public Const ERROR_NOT_BINARY_MACRO = 25693
Public Const ERROR_DCB_NOT_FOUND = 25694
Public Const ERROR_STATE_MACHINES_NOT_STARTED = 25695
Public Const ERROR_STATE_MACHINES_ALREADY_STARTED = 25696
Public Const ERROR_PARTIAL_RESPONSE_LOOPING = 25697
Public Const ERROR_UNKNOWN_RESPONSE_KEY = 25698
Public Const ERROR_RECV_BUF_FULL = 25699
Public Const ERROR_CMD_TOO_LONG = 25700
Public Const ERROR_UNSUPPORTED_BPS = 25701
Public Const ERROR_UNEXPECTED_RESPONSE = 25702
Public Const ERROR_INTERACTIVE_MODE = 25703
Public Const ERROR_BAD_CALLBACK_NUMBER = 25704
Public Const ERROR_INVALID_AUTH_STATE = 25705
Public Const ERROR_WRITING_INITBPS = 25706
Public Const ERROR_INVALID_WIN_HANDLE = 25707
Public Const ERROR_NO_PASSWORD = 25708
Public Const ERROR_NO_USERNAME = 25709
Public Const ERROR_CANNOT_START_STATE_MACHINE = 25710
Public Const ERROR_GETTING_COMMSTATE = 25711
Public Const ERROR_SETTING_COMMSTATE = 25712
Public Const ERROR_COMM_FUNCTION = 25713
Public Const ERROR_CONFIGURATION_PROBLEM = 25714
Public Const ERROR_X25_DIAGNOSTIC = 25715
Public Const ERROR_TOO_MANY_LINE_ERRORS = 25716
Public Const ERROR_OVERRUN = 25717
Public Const ERROR_ACCT_EXPIRED = 25718
Public Const ERROR_CHANGING_PASSWORD = 25719
Public Const ERROR_NO_ACTIVE_ISDN_LINES = 25720
```

```
Public Const ERROR_NO_ISDN_CHANNELS_AVAILABLE = 25721
```

```
'
' Declarations for functions to encode and decode files, typically
' used as with attachments to mail messages or news articles
'
```

```
Declare Function EncodeFile Lib "UUCODE32.DLL" Alias "EncodeFileA" (ByVal InputFile As String, ByVal
OutputFile As String) As Long
Declare Function DecodeFile Lib "UUCODE32.DLL" Alias "DecodeFileA" (ByVal InputFile As String, ByVal
OutputFile As String) As Long
Declare Function DecodeBase64File Lib "UUCODE32.DLL" Alias "DecodeBase64FileA" (ByVal InputFile As
String, ByVal OutputFile As String) As Long
Declare Function EncodeBase64File Lib "UUCODE32.DLL" Alias "EncodeBase64FileA" (ByVal InputFile As
String, ByVal OutputFile As String) As Long



Type ParseType          'user defined type used to hold
    Current As String   'the commands storing the current piece
    LeftOver As String  'as well as what is left over
End Type

'------------------------------------------------------------------------
'
'Function:  FindAnd
'
'Purpose:   Breaks up each command around the ands, and calls the
'           word parser for each one
'
'Variables: AndStuff - User defined variable holding the information
'           which needs to be parsed
'------------------------------------------------------------------------

Sub FindAnd(AndStuff As ParseType)

    Dim Pos, Leng As Long
    Dim UntilSpc As ParseType
    Dim sBuffer As String

    frmmain.txtCom = ""     'initialize the text boxes
    frmmain.txtLoc = ""
    frmmain.txtPar = ""
    frmmain.txtAtt = ""
    frmmain.txtAct = ""
    Pos = InStr(1, AndStuff.LeftOver, "and", 1) 'check for an "and"
    Leng = Len(AndStuff.LeftOver)               'store the length

    If Pos <> 0 Then                            'if and and was found then
                                                'reset the andstuff var
        AndStuff.Current = Mid(AndStuff.LeftOver, 1, Pos - 2)
        AndStuff.LeftOver = Mid(AndStuff.LeftOver, Pos + 4, Leng - Pos + 1)
                                                'itialize untilspc
        UntilSpc.Current = ""
        UntilSpc.LeftOver = AndStuff.Current
        Call ParseME(UntilSpc)
                                                'build the text box that
                                                'will be used to send the
                                                'command to the sequencer
        frmmain.txtCom = frmmain.txtCom + frmmain.txtLoc + ":"
        frmmain.txtCom = frmmain.txtCom + frmmain.txtPar + ":"
        frmmain.txtCom = frmmain.txtCom + frmmain.txtAtt + ":"
        frmmain.txtCom = frmmain.txtCom + frmmain.txtAct

        sBuffer = frmmain.txtCom.Text + Chr(13) + Chr(10)




        frmmain.Socket1.Write sBuffer, Len(sBuffer)

                                                'make a recursive call
        Call FindAnd(AndStuff)
    Else
```

```
        UntilSpc.LeftOver = AndStuff.LeftOver
        Call ParseME(UntilSpc)
        frmmain.txtCom = frmmain.txtCom + frmmain.txtLoc + ":"
        UntilSpc.Current = ""                           'initialize untilspc
        frmmain.txtCom = frmmain.txtCom + frmmain.txtPar + ":"
        frmmain.txtCom = frmmain.txtCom + frmmain.txtAtt + ":"
        frmmain.txtCom = frmmain.txtCom + frmmain.txtAct

        sBuffer = LCase(frmmain.txtCom.Text + Chr(13) + Chr(10))

        frmmain.Socket1.Write sBuffer, Len(sBuffer)

                                                'build the text box that
                                                'will be used to send the
                                                'command to the sequencer
        Exit Sub
    End If

End Sub

'---------------------------------------------------------------------------
'
'Function:  ParseControl
'
'Purpose:   Calls FindAnd with AndStuff
'
'---------------------------------------------------------------------------

Sub ParseControl(AndStuff As ParseType)

    Call FindAnd(AndStuff)

End Sub


'---------------------------------------------------------------------------
'
'Function:  ParseME
'
'Purpose:   Breaks up each command into individual words
'           Then calls checklist to see if the words are in any
'           of the lists
'
'Variables: UntilSpc - User defined variable holding the information
'           which needs to be parsed
'---------------------------------------------------------------------------

Sub ParseME(UntilSpc As ParseType)

Dim Pos, Leng As Long
Dim strTemp As String

    Pos = InStr(1, UntilSpc.LeftOver, " ", 1)   'check for a space
    Leng = Len(UntilSpc.LeftOver)               'store the length of the string
    strTemp = ""                                'initialize strTemp
    If Pos <> 0 Then                            'if a space exists
                                                'initialize untilspc
        UntilSpc.Current = Mid(UntilSpc.LeftOver, 1, Pos - 1)



        UntilSpc.LeftOver = Mid(UntilSpc.LeftOver, Pos + 1, Leng - Pos + 1)



        If (UntilSpc.Current = "measure") Then  'check for a special case
                                                'if it exists store it
            Pos = InStr(1, UntilSpc.LeftOver, " ", 1)
            If Pos = 0 Then
```

```
                End If
                strTemp = UntilSpc.Current + " " + Mid(UntilSpc.LeftOver, 1, Pos - 1)
                frmmain.txtLoc = strTemp
            End If  Pos = Len(UntilSpc.LeftOver) + 1
            If (UntilSpc.Current = "track") Then    'check for a special case
                                                    'if it exists store it
                Pos = InStr(1, UntilSpc.LeftOver, " ", 1)
                If Pos = 0 Then
                    Pos = Len(UntilSpc.LeftOver) + 1
                End If
                strTemp = UntilSpc.Current + " " + Mid(UntilSpc.LeftOver, 1, Pos - 1)
                frmmain.txtPar = strTemp
            End If
            If (strTemp = "") Then                  'if no special case was found
                                                    'call checklist
                Call frmmain.CheckList(UntilSpc.Current)
            End If
            Call ParseME(UntilSpc)                  'recurse ParseME
        Else
                                                    'call checklist
            Call frmmain.CheckList(UntilSpc.LeftOver)
            Exit Sub
        End If

    End Sub
```