Integrating Speech Recognition and Generation Capabilities Into Timeliner

by

James M. Napier

Submitted to the Department of Electrical Engineering and Computer Science in Partial Fulfillment of the Requirements for the Degree of Bachelor of Science in Computer Science and Engineering and Master of Engineering in Electrical Engineering and Computer Science at the Massachusetts Institute of Technology

May 24, 1996

Author ....[illegible].......
Department of Electrical Engineering and Computer Science
May 24, 1996

Certified by ....[illegible].......
Richard Berthold
VI-A Company Supervisor

Certified by ....[illegible].......
Stephanie Seneff
Thesis Supervisor

Accepted ....[illegible].......
F. R. Morgenthaler
Chairman, Department Committee on Graduate Theses

Integrating Speech Recognition and Generation Capabilities into Timeliner

by
James M. Napier

Submitted to the Department of Electrical Engineering and Computer Science

May 24, 1996

In Partial Fulfillment of the Requirements for the Degree of Bachelor of Science in Computer Science and Engineering and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

Timeliner is a control sequence language designed for real-time script operation. It is designed to be used on board the International Space Station allowing the crew members to control script operation with relative ease. The addition of speech recognition and generation capabilities into the Timeliner interface further increases the user friendliness by allowing crew members to be freed from the constraints of a mouse driven X window interface. The recognition and generation modules show the extensibility of the current interface and provide a proof-of-concept basis for further study into the functionality gained through a speech interface.

Thesis Supervisor: Stephanie Seneff
Title: Principal Research Associate, MIT Laboratory for Computer Science

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Thesis Overview

The goal of this thesis is to show the feasibility and usefulness of a speech interface with Timeliner--a control sequence language designed for the International Space Station. The addition of a speech interface for Timeliner will provide the crew with a greater ability to control the operation of Timeliner scripts being run on board the station. The speech interface should include both a speech recognition module to interpret the crew's verbal commands and a speech generation module to audibly relay messages from Timeliner scripts.

This chapter discusses the goals of the research and the organization of the thesis. It gives an overview of Timeliner and of the necessities of a speech interface for Timeliner.

1.2 Timeliner Overview

Timeliner is a language designed at the Charles Stark Draper Laboratory under contract from the National Aeronautics and Space Administration(NASA)[1] and the Boeing Corporation. The goal is to be able to handle the scheduling and monitoring of events on board the International Space Station. This could include both station operation and scientific experiments performed on the station.

Timeliner is designed to alleviate the demands placed on the crew by mechanizing certain aspects of their daily workload. The shrinking of funds for research in space has forced projects such as the International Space Station to bear an even greater load by trying to maximize the use of the station. This corresponds to a greater workload for the crew.

1. NASA Contract No. NAS 9-18426 Task Order Control 95-123

Timeliner provides a way to automate various parts of experiments while still allowing the crew to interact with and control the operation of these scripts. The scripts can be set up to run autonomously or with crew interaction. Timeliner provides facilities for doing this. With Timeliner, the experiments can be more easily maintained and can be parallelized.

The Timeliner language is designed for writing sequencing procedures to operate complex systems. It can handle multiple tasks in real-time. The language itself is very English-like allowing easy comprehension by non-programmers. The Timeliner language is developed using the Ada programming language [1] which provides facilities for real-time tasks and data abstraction.

The following table is a simple Timeliner script which shows the English-like and easy-to-understand syntax [2].

Table 1.1: Simple Timeliner Script

```
--This is a simple Timeliner script which waits on an action and then sets a variable

BUNDLE Test_Experiment
    SEQUENCE Main ACTIVE
        START Monitor
        WHENEVER Pump_Speed > 10 THEN
            SET Valve OPEN
        END WHENEVER
    CLOSE SEQUENCE Main
    SEQUENCE Monitor INACTIVE
        EVERY 5.0 BEFORE Experiment_Off
            IF Pump_On THEN
                SET Pump_Speed TO Pump_Speed + 1
            END IF
        END EVERY
    CLOSE SEQUENCE Monitor
CLOSE BUNDLE Test_Experiment
```

It starts up a sequence, Main, which opens a valve when the pump speed is greater than 10. In parallel with this operation, the sequence Monitor gets started which increments the value of the pump speed as long as the experiment off variable is false. It is very easy to understand since it reads like a check-list.

In order to make the script operation easily controlled, Timeliner supports an X-window interface. Through a variety of panel displays (See "Timeliner Windows" on page 45), the operation of every script being executed by Timeliner may be viewed and controlled.

1.3 Station Operation

Since the station requires that various operations be performed to control station operation, Timeliner is a tool by which those operations may be automated. A script does not necessarily require crew participation in order to function. Therefore, certain scripts can be used to control basic tasks, such as temperature control of the station modules.

Another main aspect of the station is to provide a platform for performing the myriad of microgravity experiments that have been conceived. Through a script, an experiment can be controlled with Timeliner, and the crew can be easily involved with the experiment via the X-window interface.

The simple-to-understand scripts facilitate the correct running of experiments. It is very costly if an experiment is done incorrectly. Timeliner provides an extra measure of safety because the crew can easily follow the script operation. Also, the script can signal them to perform certain operations while the experiment is running.

This interactivity of Timeliner is a major advantage for performing experiments on board the station. Certain operations cannot be sufficiently automated and still require human intervention. Also, some experiments are quite complex. Timeliner's scripting language provides a mechanism for the scientists who created the experiment to easily describe experimental parameters. This allows for a greater chance of successful completion of the experiment.

1.4 Enhanced Interface

The current interface requires the crew to be at a display in order to interact with the scripts being run. Mouse inputs are needed to control the script operation and any messages that are sent from Timeliner are displayed to the screen, requiring the crew to be able to view the screen in order to see the messages. A speech interface would alleviate some of these constraints.

To allow the crew to be mobile, the first step is to provide a mechanism so the crew need not use mouse inputs to control the operation of Timeliner. A speech recognition module could fulfill this role. Any commands done using a mouse would correspond to verbal commands given through a headworn microphone. The second step is to allow the messages that are sent to the screen to also be routed to a speech generation module so the crew could hear the messages through a headset. This would free the crew to perform the experiment while still receiving feedback from the script operation. Sound has the significant advantage that it does not require direct focus of attention on the part of the crew in order to receive the information.

A speech interface also allows for quicker control by the crew since they no longer need to "float" over to the displays to direct Timeliner. The addition of speech will provide multiple avenues of feedback so the crew can use whichever is more convenient; and the information provided is now more understandable. The redundancy of the information is an asset because it utilizes both sight and sound instead of merely sight.

1.5 Considerations for the Interface

The speech interface has some important aspects which affect the final product. The litmus test is how the interface appears to the crew and its ease of use. The additional speech interface should be relatively transparent to the crew. While the speech interface is integrated to some degree, it should never compromise the current interface usability. It should allow them to better understand the script operation and allow them to do their jobs more efficiently and with less stress. This would also empower script writers to further aid the crew in understanding how an experiment works. Additional messages might be warranted if the verbal messages would enhance the functioning of the crew.

1.6 Summary

In this chapter, we have shown why Timeliner helps the crew accomplish its mission, specifically in performing experiments. We discussed why a speech interface would further promote the purpose of Timeliner with added functionality. The crew would be given a better tool to accomplish their tasks. The next chapter covers the steps we took to define the requirements of a speech interface for Timeliner.

Chapter 2

Project Scope

This chapter discusses the requirements that a speech interface must meet for Timeliner. It also introduces the software packages that are used to create the speech interface.

2.1 Requirements

The speech interface needs to be modular with respect to the rest of Timeliner. Not only does the problem lend itself to modularity, but it is a good practice since some versions of Timeliner might not require the use of a speech interface. It should also be relatively lightweight with regards to size and speed. While functional versions of Timeliner are run on workstations, the flight version must use station computers, which are Intel™ 386 processor-based. It might be desirable, for example, to completely replace either the recognizer or generator with a different package. With a good interface, such a process would be trivial.

In addition to being lightweight, the speech interface should be easily maintained. This includes the ability to change the interface with relative ease. It should be possible to add speech recognition commands quickly and simply. Considering the application, there will be a limited number of users and a relatively small vocabulary. Therefore, it is conceivable to use a system that requires training, but the training should be straightforward. A serious issue is the recognition performance. It should provide good recognition in the face of noise. To this end, a microphone of good caliber should be used to help reduce the signal-to-noise ratio.

The timing of the speech interface should be relatively quick. While the speech recognition will happen in parallel with the script operation, it should also appear to have an effect as soon as possible to reflect the changes made by the verbal commands. Further-more, the generated speech must be comprehensible, and it would be advantageous if it were customizable to the crew's preferences to some degree.

It should be noted that this thesis deals with a functional version of Timeliner, which runs on a Sun™ Sparc10 workstation. This thesis is centered around a proof-of-concept for the idea of a speech interface for Timeliner. This results in some additional constraints on the speech interface modules. The software must run under the correct operating system and the purchase of any items used for the thesis must be cost-effective.

2.2 Commercial Versus Internally Developed Software

The end goal of this research is to show the feasibility of a speech interface for Timeliner. Commercial speech recognition and generation packages provide a viable option for use with this thesis. Additionally, the Timeliner Project would like to continue with this research after this thesis is done, and commercial software allows for continued support for the features desired. There is no need to reinvent the wheel in order to accomplish the goal of this research.

2.3 Commercial Speech Recognition

There are various types of speech recognition algorithms used in different commercial packages. Each provides trade-offs between the recognition rate, the size of the vocabulary, and the need for training. Some packages require discrete speech, while others allow for continuous speech input. Each of these attributes' importance depends on the application of the recognizer. In this case, the International Space Station and Timeliner provide a specific environment in which the recognition must perform.

The recognition in this environment must be extremely accurate. It would be very confusing if the recognition failed to recognize a spoken command, but it would be even worse if it misinterpreted a spoken command. Also, the station will most likely have a relatively high amount of noise due to computer fans and environment control devices. The recognition needs to provide accurate results in spite of a noisy environment.

The speech recognition is limited to the number of commands that the displays provide. This is on the order of about 100 commands. This means that the recognition package need not have a huge vocabulary. More importantly, however, it means that the recognition package could use training in order to provide recognition. This holds true because there will be a limited number of crew members, so the training could occur with each crew member. Remember, though, that the training should be relatively easy and quick in order to be usable.

The requirement about discrete speech versus continuous is rather flexible. It would be nice if the recognition could handle continuous speech, but the crew could just as easily deal with a package that requires discrete speech. The only constraint is that the crew should not have to enable the recognition with anything other than a voice command (*e.g.* depressing a button to initiate recording).

2.4 Comparison of Speech Recognition Packages

The following table shows the comparison of some commercial speech recognition packages that are available for the Sun Sparc10™ running Solaris®2.5.

Table 2.1: Speech Recognition Packages

Package	Recognition Model	Training Required	Discrete vs. Continuous	Cost
Abbot	Connectionist Hidden Markov Model	No	Continuous	free
IN^3	Energy Template	Yes	Continuous	$500
Lotec	Templates and Word Hypothesis	Yes	Continuous	free
Hark	Hidden Markov Model	No	Continuous	Expensive (depends on vocabulary size)

2.5 IN^3 Speech Recognition

Based on all the factors related to the recognition environment, IN^3 (pronounced in-cube) seems to be the best choice [3]. It provides a mechanism for easily training the verbal commands and it has very good recognition. It uses continuous word-spotting algorithms based on an energy template, or voice pattern, of the verbal commands. A command can be recognized within a verbal utterance without requiring that the command itself be bounded by silence.

The user interface is very intuitive and, therefore, commands can be easily added. The software provides multiple ways to add context sensitivity to the recognition. This means that certain commands should only be recognized when a certain condition exists (the crew is focused in a specific window, for example). Through the concept of contexts, IN^3 constrains its pattern recognition to only a subset of possible verbal commands. This greatly increases the recognition rate and decreases the occurrence of incorrect recognition.

Since one of the goals is to mimic mouse input to the displays, it is necessary to be able to control X-events from the recognition of the verbal commands. IN^3 has the ability to capture X-events when creating commands and then perform those X-events when it recognizes the corresponding command. Another great aspect is that the software can appear transparent by loading up automatically with the correct verbal templates and with the recognition active. However, the recognition parameters can be changed by bringing up a simple user interface. IN^3 is very flexible and robust. It is integrated well with X-windows and is relatively cost-effective.

2.6 Speech Generation

The speech generation is relatively straightforward with regards to its constraints. It should be able to take a string of text and convert it to audible speech in a comprehensible manner. Since this environment will be routing text messages to the speech generation module from Timeliner, the module should operate on text and output good comprehensible speech.

The following table shows some speech generation packages and the important attributes associated with each.

Table 2.2: Speech Generation Packages

Generation Package	Text-driven	Output Quality	Cost
rsynth	Yes	poor	free
Entropic's Personal TrueTalk™	Yes	good	$500
YorkTalk	No	fair	Unknown

2.7 Personal TrueTalk Speech Generation

Entropic's Personal TrueTalk™ generation package provides a very good interface for taking text from various sources and converting it to speech [4]. The software was developed at Bell Laboratory which is known for good speech generation software. The output is extremely comprehensible and has very good intonation in a sentence. The quality is well worth the cost. Also, the software provides a few ways to send the speech to the generation engine--TCP/IP ports, Tcl/TK routines, and by typing the text in an X-window. The X-window interface allows a very simple way to test the speech output quality for quick development. And since the final result must look transparent, the window need not be used, because the text can be sent via a Tcl script. It runs well on the workstation and also provides ways to change the output speech according to user preferences.

2.8 Microphone

The microphone on a regular workstation is an omni-directional microphone, which introduces all the background noise of the environment in which it is being used. Therefore, it is necessary to acquire a better microphone in order to increase the speech recognition rate by reducing the signal-to-noise ratio. A good directional headworn microphone costs $200, but the sound quality is much improved. Also, the headworn microphone mimics what the International Space Station crew really has. It frees up the hands so that other things can be done and lowers the chance of interference from other talkers who may be present.

2.9 Integrating

This chapter has focused on the necessary requirements that a speech interface must fulfill to be used with Timeliner. It also introduced the two software packages that are used to create the speech interface, IN^3 and Personal TrueTalk. The use of the speech recognition software and the speech generation software allow for specific modules to be hooked into Timeliner to create the speech interface. Therefore, the remainder of the thesis focuses on the integration of the speech recognition and generation modules and on the issues associated with a speech interface for Timeliner.

Chapter 3

Implementation

This chapter shows how the speech interface is created. It gives further insight to the organization of commands into contexts, patterned after the Timeliner displays. Finally, it introduces a Timeliner script that shows the utility of a speech interface for Timeliner.

3.1 Organization of Speech Recognition

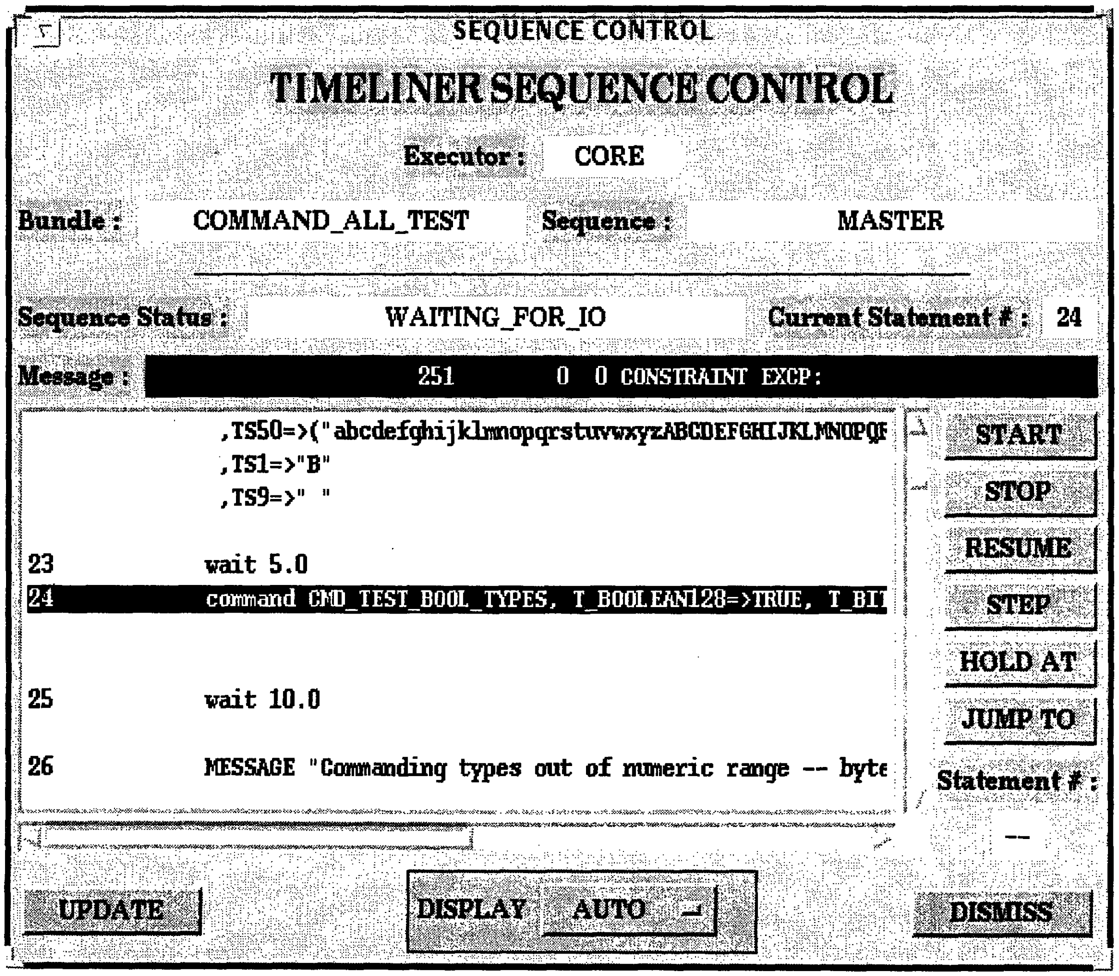
The speech recognition should mimic the possible mouse inputs to the display. Each display has a set of possible commands that can be accomplished through mouse clicks. In order to increase the recognition rate, each display has a certain set of verbal commands that can be recognized while within that display's context. Therefore if a new window comes up within Timeliner, a new context is activated. The contexts based on windows create a hierarchy of commands that are recognized. There are also commands which allow the crew to navigate between the windows and scroll within windows.

This chapter deals with the way in which the speech recognition and generation are connected to Timeliner. The creation and training of commands and the routing of messages to the generation package are discussed. Finally, we introduce a demonstration Timeliner script that we created to show the value of a speech interface with Timeliner.

3.2 Timeliner Displays

There are approximately nine windows in the Timeliner X-window interface. Each one corresponds to a different context for the speech recognition. The buttons on each allow various access to the other windows in the interface as well as control over the scripts being run. This is the only way a crew member can get status, currently. The speech interface extends this by allowing the crew to hear messages and to control the windows with voice commands.

The following Timeliner display is an example (See "Timeliner Windows" on page 45). There are verbal commands associated with each button and scroll bar that can be depressed. The buttons provide the crew with access and control over the sequence that is running. Different sequences can be loaded up and viewed even while other sequences are running. The recognition must be able to differentiate between the "START," "STEP," and "STOP" commands, which are all Timeliner commands. Also, the mouse navigation controls for the scroll bars and for selecting lines in the sequence correspond to "SCROLL" commands and "SELECT NEXT" and "SELECT PREVIOUS" commands, respectively.



TIMELINER SEQUENCE CONTROL

Executor : CORE

Bundle : COMMAND_ALL_TEST Sequence : MASTER

Sequence Status : WAITING_FOR_IO Current Statement # : 24

Message : 251 0 0 CONSTRAINT EXCP:

```
          ,TS50=>("abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQF
          ,TS1=>"B"
          ,TS9=>" "

23        wait 5.0
24        command CMD_TEST_BOOL_TYPES, T_BOOLEAN128=>TRUE, T_BI

25        wait 10.0

26        MESSAGE "Commanding types out of numeric range -- byte
```

Statement # : --

Figure 3.1: Sequence Control Window

3.3 Mimicing Mouse Clicks

The ability to mimic mouse actions is not straightforward. There are certain advantages to mouse movements that do not easily translate into verbal commands. Therefore, care must be taken to insure that as much of the current interface is extended to verbal commands as possible. One advantage of verbal commands is that their utterance can be closely tied to the action which they perform. For instance, most buttons of the Timeliner window interface have a corresponding verbal utterance that is the same as the text label of the button (See "List of Commands" on page 54).

Some mouse movements are not associated with specific buttons--window focus for example. In order to switch between windows, there are verbal commands that reference specific windows. This provides a mechanism by which different classes of verbal commands can be loaded depending upon the current active window.

3.4 Verbal Command Contexts

The contexts of each window differ, thereby creating subsets of verbal commands for the recognition package. This is needed because sometimes the same verbal command should do something different depending on the window that is active (the context). For example, the command "DISMISS" is used in many windows to cause the window to disappear, but the actual X-event that takes place differs for each one. If there were no mechanism to allow for contexts, then each "DISMISS" would have to be unique in its verbal command template.

Each command need only be recognized when the user is activating the portion of the Timeliner interface that is related to the command. By keeping the commands separated into a hierarchy of classes, the number of commands that the recognition engine must correctly recognize is certainly reduced, thereby increasing the recognition rate. To augment this hierarchy, there must be a manner to maneuver between states of the command hierarchy. This results in a state diagram that provides connectivity while maintaining local minimization of the number of active commands. The following figure shows the state diagram of all the commands and the contexts in which they are active.

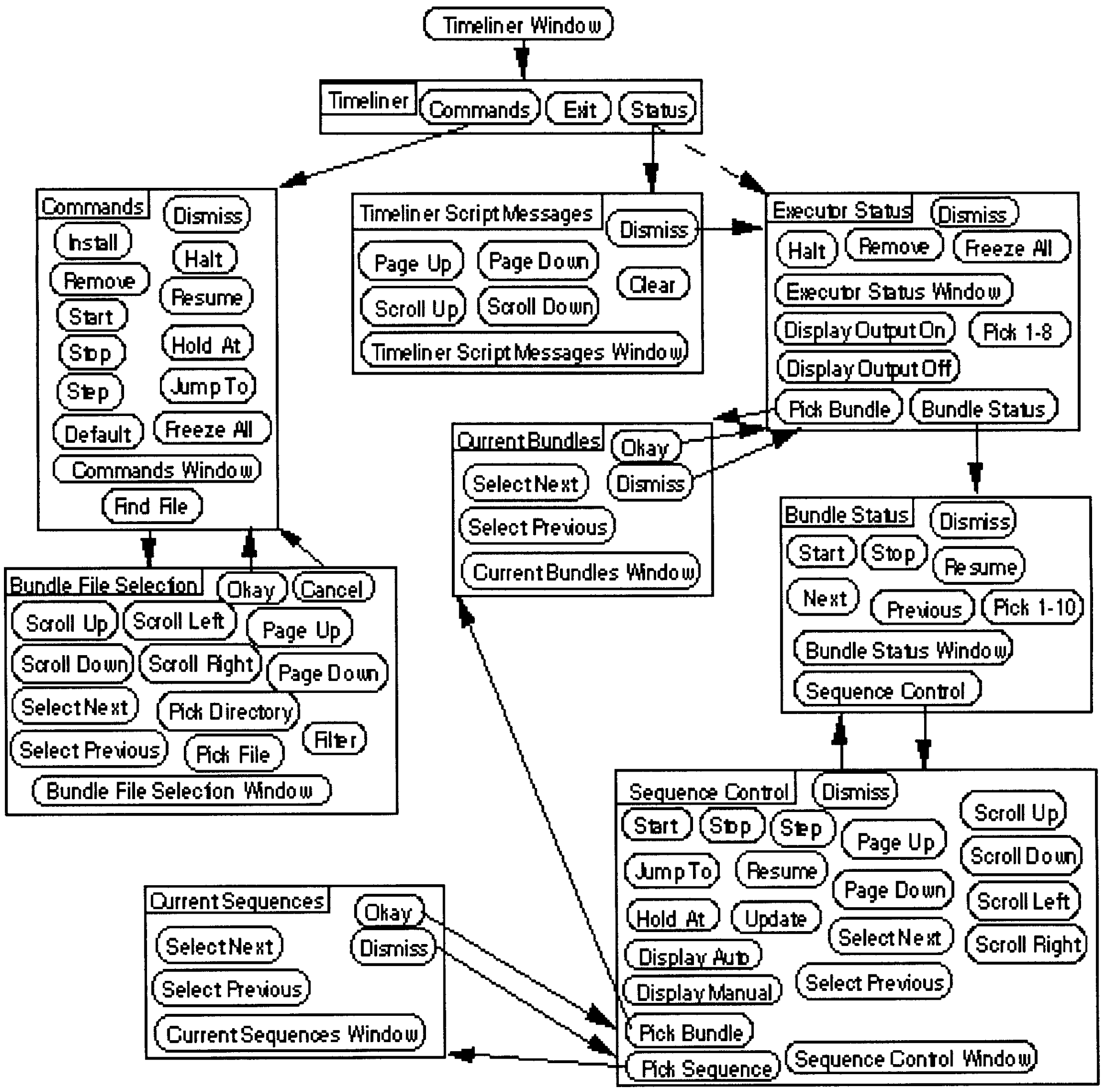


Figure 3.2: Context State Diagram of Spoken Commands

Notice that there must be a class of commands which are always recognized. This allows for commands to switch between the different windows, hence the different contexts.

IN^3 has two different ways to recognize contexts within an application. The first is a field related to each command that maintains the X-event class name in which the command should be activated. While this sounded promising, it proved to be of little use since the class names within the same application are the same. The second method uses a concept of command sets. Each command is part of at least one command set, represented by a bit mask. Therefore, there can be commands that are a part of multiple command sets, allowing some commands to be activated within multiple contexts.

At any one time, at most two command sets are active, one of which is always the default. The default command set is called "NEVER_OFF" and any commands contained within can always be recognized. The following table lists every command set, the window with which it is associated, and the number of commands in each set.

Table 3.1: Command Sets

Command Set	Associated Window	Number of Commands
Bundle Status	Bundle Status Window	19
Commands	Commands Window	14
Current Bundles	Current Bundles Window	5
Current Sequences	Current Sequences Window	5
Executor Status	Executor Status Window	16
Find File	Bundle File Selection Window	14
Never Off	not applicable	10
Sequence Control	Sequence Control Window	21
Timeliner	Timeliner Window	12
Timeliner Script Messages	Timeliner Script Messages Window	7

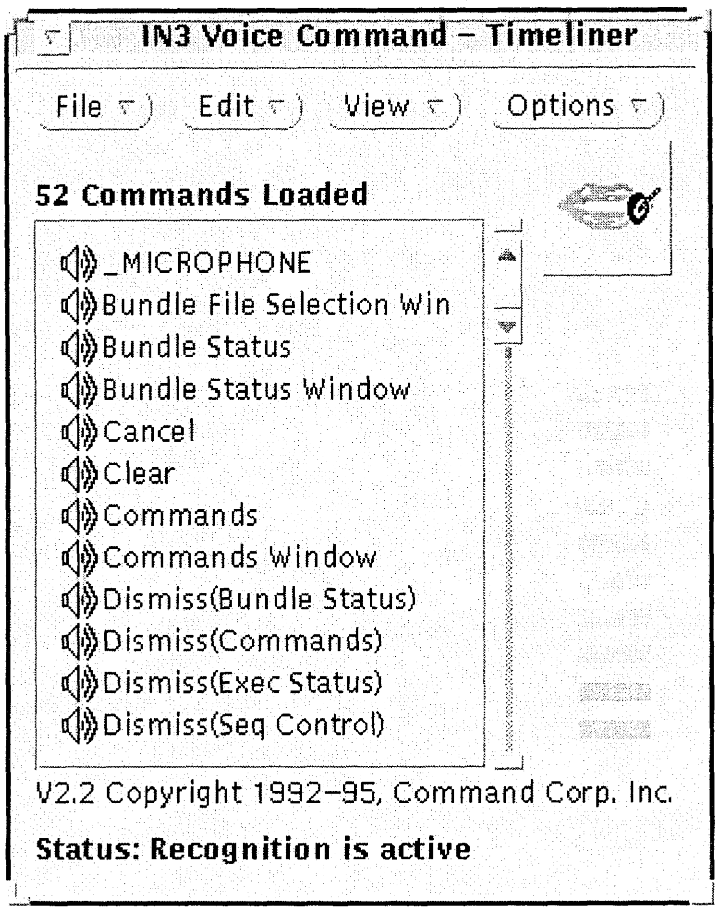
The total number of distinct commands currently implemented is 100. From the above table, it is obvious that the use of contexts severely cuts down the number of commands that the recognizer must deal with at any one time (by more than a factor of five).

An interesting thing to note concerns incorrect recognition in this framework. The effect of incorrect recognition is severely limited through the use of contexts. By carefully choosing the verbal commands within a context, false recognition can usually be avoided. Also, since the command sets are associated with certain windows, false recognition can be limited to harmless functions. In order to get access to the potentially undesirable commands of a non-active window, it is first necessary to say a command which moves the focus from the current window to the unwanted window. Luckily, when the command to switch to a new window is recognized, there are multiple indications that such a situation has occurred. First, the window related to the new context is suddenly brought to the front of the display if it is behind other windows. Then, IN^3 has a setting that causes an audible beep to occur that indicates that a new context is active. With this feedback, it becomes apparent that a new context/window has been activated. Of course, this is usually unnecessary as long as the verbal commands are chosen wisely.

3.5 Enabling IN^3

In order to make the speech interface as modular as possible, IN^3 is started up as its own process rather than being a necessary component of the Timeliner baseline. In this way, the recognition is not critical to the operation of Timeliner, yet still provides the functionality of an integrated package. In fact, it is possible to create voice commands that load up and run the necessary components to initiate Timeliner. While not necessary for a final version of the commands, this feature is very valuable when designing the commands, because Timeliner can be quickly restarted when problems arise during the initial design of new verbal commands.

The default for IN^3 is to load up a window interface that provides many tools for developing new commands and using them. The following figure shows the main window for IN^3.



52 Commands Loaded

_MICROPHONE
Bundle File Selection Win
Bundle Status
Bundle Status Window
Cancel
Clear
Commands
Commands Window
Dismiss(Bundle Status)
Dismiss(Commands)
Dismiss(Exec Status)
Dismiss(Seq Control)

V2.2 Copyright 1992–95, Command Corp. Inc.

Status: Recognition is active

Figure 3.3: IN^3 Window

This window is very useful when developing and testing new commands. It gives feedback about what commands get recognized and which commands have voice templates associated with them. From this window, menus allow the commands to be created, trained, and refined. However, during a full integration with Timeliner, IN^3 need not be started with a window interface. Therefore, the speech recognition could appear completely transparent to the crew members on board the station.

3.6 Creating Commands

Commands are created using a window interface with IN^3. First information needs to be created identifying the command, its operation, and its context. This is accomplished in the command window of IN^3 (See "IN3 Command Edit Window" on page 53).

3.7 Speech Commands for Timeliner

The commands for Timeliner are easily created using the interface for IN^3. Each command allows for window navigation or mouse clicking associated with the Timeliner interface. The commands are simple to say and easy to remember since their function either corresponds directly to buttons with labels or are intuitive scrolling functions with names such as “SCROLL UP” and “PAGE DOWN” (See "List of Commands" on page 54).

3.8 Training the Commands

In order for the speech engine to recognize a command in IN^3, each command must be associated with a template. The template is a recording of the time frequency energy pattern of the spoken command by a particular speaker. IN^3 allows commands to be duplicated, with different verbal templates associated with the duplication. It is even possible to have the commands with different templates perform the same action (one could have “CLOSE WINDOW” and “DISMISS” do the same thing). Using this, each crew member has a unique template for the commands that the crew member can say.

The energy patterns associated with each command allow for variations in speech rate and pitch. It is important to note that this means that each verbal command can be recognized correctly when spoken by someone else besides the person on whom it was trained. Thus, there is very little guarantee of authentication on the part of the speaker (See "Further Research" on page 40).

The commands are trained by a simple creation using IN^3's interface. Each command being trained is repeated twice before moving on to the next command. Then, all the new commands are repeated one more time. This minimizes the problems that occur when users change their speech pattern when training the commands as compared to normal operation. If such a change occurs, then the recognition rate suffers. By having to speak them, one after the other, the user unconsciously reverts to a normal speech pattern, increasing the likelihood of correct recognition.

IN^3 also allows a command to be refined. This is a similar operation, but it merely adjusts the previous template with the newer recording of the energy pattern. Since the creation routine is very simple for commands, it might be easier just to recreate the commands if there is any doubt about a template's correctness. Considering the number of commands and the number of users of this recognition system, it is not unreasonable to have to train the recognition. This cuts down greatly on the restrictions of the speech recognition task with an acceptable trade-off of a relatively small increase in training time.

3.9 Organization of Speech Generation

The generated speech should mimic the messages that are displayed on the screen. It should provide a simple way for the crew to learn of any change in status of the current scripts which are running. This provides a distinct advantage over a textual display of the message to the Timeliner window interface. Most importantly, the messages will be easily distinguishable from other feedback to the crew. The speech generation has the possibility to distill the plethora of information that the Timeliner script contains into a simple audible message of status. Of course, a primary requirement is for the speech to be comprehensible. Therefore, the messages might need to be reformatted to insure the best possible form for the generation engine.

The generation package should be as modular as possible since it only relies on text strings from Timeliner scripts. Since the messages are already routed to the Timeliner displays, it is from this point in the code that the messages can also be sent to the speech generation engine. One thing to aim for is fault tolerance. It would be unwise to expect Timeliner to wait for the generation engine to complete, since messages could be arbitrarily large. Instead, the messages should be sent to a separate process which takes care of the speech generation package, thereby insuring that Timeliner can continue.

3.10 Enabling Personal TrueTalk

The speech generation software, Personal TrueTalk, normally loads up as a window interface. This window can be closed while still allowing speech generation to continue. The window allows the user to adjust various settings such as voice gender, voice rate, and volume level. Also, part of the window is an area for trying out sentences so that the user can get an idea of how the speech generation will sound. The following figure shows the main window for TrueTalk.

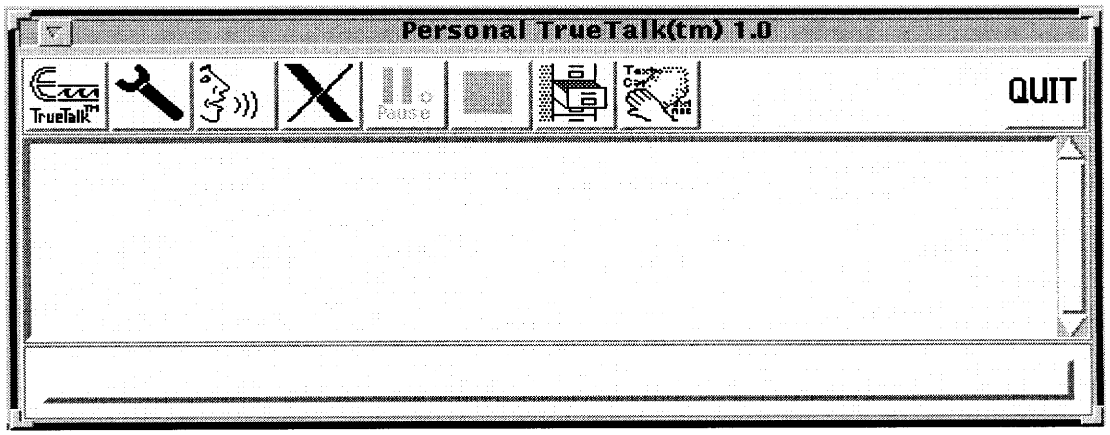


Figure 3.4: TrueTalk Window

TrueTalk provides multiple ways other than typing into the window in order to pass text to the generation engine. One manner is the ability to speak highlighted X-selections. Another is through the use of TCP/IP ports. The text can also be passed using Tcl/TK commands. The TCP/IP port method likely provides the most secure method for passing text to TrueTalk since TCP is a reliable protocol. However, one problem that can arise is that causality is not necessarily maintained. If one text message gets delayed due to a failure in sending and another process has begun to speak a new selection, there is no inherent guarantee as to which will arrive first. In all fairness, however, a mechanism could be implemented which strictly enforces causality. However, it is preferable if an elaborate control system can be avoided while still achieving causality.

As it turns out, the use of Tcl/TK commands can maintain causality without complex control. TrueTalk has an added TK command called "TTplay" which speaks strings that are passed to TrueTalk. By using this, it is simple to create a Tcl script that calls a generation package with a "send" command.

The generation package is called from a Timeliner call when the message is output to a display. This is done with only one procedure called "speak_TL_message" in the Ada package "TL_speech_gen_interface" (See "Ada Code for Speech Generation" on page 59). It is called and given the message that should be spoken as a string.

In order to interface the Ada code with a Tcl/TK script, an intermediate C program is used (See "C code for Speech Generation" on page 61) [5]. The Ada procedure prepares the string and calls another procedure which is actually the C code.

The C code starts up a process of a Tcl interpreter with a Tcl script. The Tcl script is responsible for making the actual TK "send" call to the generation engine (See "Tcl/TK Scripts" on page 62) [6]. One problem that arose is related to the time it takes to actually speak a message. Sometimes in Timeliner scripts, a few messages are quickly displayed one after another. Therefore, care is taken to insure that any new messages wait until the engine is not busy before proceeding. To recognize that the generation engine was busy speaking a command, another “send” had to be devised to get access to a variable called “TTPlaying”. One problem with this call is that “TTPlaying” does not exist until the first time that the engine attempts to speak some text. Therefore, two Tcl scripts are needed. The first Tcl script assumes that the engine is not busy and calls it without checking that it is busy. This Tcl script is only executed the first time the C procedure is called. Then, any time another message needs to be spoken, a different Tcl script gets called, which makes the check on the variable and waits until the generation engine is no longer busy.

This method also doesn’t drop messages. It is bad to receive every other message, for example. However, it is unclear whether there should be some mechanism which drops messages after a certain period of time so that extremely old messages do not get spoken (See "Generation Strengths" on page 38).

3.11 Complexity

A guiding principle behind the design was Occam’s razor which states that simplicity is better than complexity. In cases where it was unclear for certain design issues, such as using Tcl/TK instead of TCP/IP, this principle was applied. While maintaining simplicity, this implementation does not sacrifice value for the speech interface.

3.12 Demonstration Script

In order to show the usefulness of the current speech interface implementation, a demonstration script was created (See "Demonstration Script" on page 65). The data used to create the demonstration Timeliner script is a real space station payload experiment from NASA’s Marshall Space Flight Center in Huntsville, Alabama. An important aspect of this demonstration script is the interaction required from Timeliner and the crew in order to complete the experiment. Each aspect of the experiment asks the crew to do certain tasks before continuing the experiment. This use of Timeliner helps the crew by making sure that each item on the experiment's checklist is done (and in the proper order).

This script shows the usefulness of the speech interface to aid the crew in understanding script operation. The messages are spoken as well as displayed and the crew is signalled when their interaction is required. Also, the crew are allowed to leave the terminal, in order to do parts of experiment, while still staying in contact with the script operation. This insures that messages are not mistakenly overlooked because the crew is not watching the displays at every moment. Also, it relieves the crew from having to go back over to where the displays are located in order to signal Timeliner to continue once the crew completes the manual operation needed for the experiment. This enhances the current interface with additional functionality.

3.13 Summary

This chapter discussed the actual implementation of a speech interface for Timeliner. We showed that the use of contexts within verbal command recognition significantly lowers the number of commands that are active for the recognizer. We also demonstrated the advantages of speech generation and how we interfaced with the generation from the Timeliner displays. The demonstration script illustrates the benefits of a speech interface for Timeliner by revealing how it can be used to help the crew. The next chapter discusses our results for this research.

Chapter 4

Results

This chapter gives the results of our research. It illustrates the effectiveness of the recognition software and the generation software to further enhance the Timeliner interface.

4.1 Does it Work?

Obviously, the most important litmus test for the speech interface is whether it works. Each separate package integrates with Timeliner perfectly. However, we had a problem with the workstation we were using which makes it very difficult for both the recognition and the generation to take place at the same time. This is because both packages must use the /dev/audio device. Since our workstation currently has a speakerbox, which hooks up to the microphone, only one package can gain access to the audio devices at a time. It turns out that there is a way to solve this problem by each package following a protocol of surrendering the audio device when another process asks for it. Unfortunately, TrueTalk does not follow this protocol. We were forced to change the code to stop recognition while a message is spoken, but it would be better if that were not necessary. It turns out to be very annoying not to be able to speak while something is being spoken. In fact, the current version can wait up to six seconds to reenable recognition after something gets spoken in the worst scenario. However, in a real experiment on the space station, it is likely that the crew would have operations to perform during the delay.

Other than that problem, the speech recognition and generation both complement the windows interface of Timeliner with the functionality of a speech interface. The interface is transparent to the user. This means that when they are running, it is not evident that the actual recognition and generation software are not part of the Timeliner baseline. Yet during creation and testing, the window interfaces of each software package allow for separate testing and evaluation of the speech interface.

4.2 Recognition Ability

The speech recognition of IN^3 is very reliable and predictable. The recognition normally is very accurate. The recognition rate is very dependent upon the environment in which the training takes place. The errors in recognition usually involve similar commands. This should be avoided if possible. However, most of the problems associated with the recognition are correctable.

The codec used in the current model of the speakerbox in the Sparc10 being used has problems with a background hiss that gets recorded along with the microphone input. This causes the automatic gain adjustment feature of the recognition software to be set much too high, thereby lowering the effectiveness of the recognition software. After consulting with the company that created IN^3, adjustments were made to minimize the effect of this deficiency in the speaker.

4.3 Generation Ability

The speech generation works extremely well. It provides a very comprehensible output of the Timeliner messages in a timely manner and in a causal fashion.

There is still an open-ended question concerning the treatment of acronyms, however. Acronyms provide a serious challenge to speech generators because the acronyms are not usually real words. In addition, some acronyms, when spoken, are changed by adding vowels in order to make the acronyms sound like a word. Currently, TrueTalk merely spells out the acronyms. This seems very reasonable, especially considering the learning curve required by humans to correctly deal with acronyms. However, it is likely that a table could be created that relates common acronyms to their phonetic interpretation and then the generation engine could consult such a table.

Messages could be better designed if the creator has the foreknowledge that the messages are going to be sent to a speech generator. This would cause the designer to pay closer attention to the messages being created. In fact, in the future, multiple messages could be maintained; one intended for screen display, and another which is phonetically engineered for specific input to a generator.

4.4 Relation to Current Interface

The speech interface enhances the current window interface with obvious additional functionality. It empowers the crew by allowing mobility. It provides further feedback to the crew members through voice messages and it provides the crew with the ability to control Timeliner operation with easy voice commands. The utility of the speech interface reveals itself in any script that gets run. The demonstration script (See "Timeliner Script" on page 65) shows how useful it is to be able to interact with the script while still maintaining control over the operation with better efficiency. This demonstration script does not show every possible use of the new speech interface. By having such an interface for Timeliner, the creators of the Timeliner scripts can probably design even better uses of the speech recognition and generation features.

4.5 Summary

This chapter revealed the results of a speech interface for Timeliner. It analyzed the advantages that the additional interface has for the crew and for Timeliner. The final chapter draws conclusions about the research.

Chapter 5

Conclusions

This chapter discusses the strengths and weaknesses of the current implementation. It also goes into the open issues that we discovered and elucidates further research that could be done, as well as the extensions that could be added.

5.1 Recognition Strengths

The speech recognition provides a useful interface for the crew. The recognition can be very convenient to use in most respects. It would not be suitable for entering keyboard inputs but it does an admirable job of saving mouse strokes. The recognition proves to be very accurate for medium sized phrases with rare mistakes. Also, the verbal utterances can be within longer phrases and the software is still able to recognize the command. This means that the crew need not maintain a regimented form for speaking the verbal commands as long as the verbal commands, themselves, are relatively similar to the templates.

5.2 Recognition Weaknesses

There are some problems with the recognition software, but most are ones that can be avoided with some precautions. One must be careful with the choice of voice commands within overlapped contexts. If the commands are too similar, then it is unlikely that the software will be able to recognize all the commands correctly. However, one factor that really attributes to this is the signal-to-noise ratio in the environment. In our environment, we had some background noise, but it turned out that the majority of noise was introduced by a poor speaker input to the workstation being used. This problem is quickly remedied by insuring a good input to the recognition software.

A simple rule to follow is to train the recognition software in the environment in which the recognition will take place. This seems to present a problem for Timeliner's intended use--on the space station--but there will be very accurate station simulators on the ground that will provide a close replica of the station itself, including background noise. Therefore, the training of the verbal commands can take place with the crew in a station simulator.

5.3 Open Issues for Recognition

Some of the problems we encountered had no clear solutions. We implemented the software on a relatively powerful workstation. On the space station, the computers will be 386 cpu-based. This presents a rather drastic drop in performance. There are speech recognition packages that are designed for such computers but it remains to be seen whether the quality can match what was demonstrated by this research. Also, not all of the mouse operations so easily lend themselves to voice commands. Picking an item from a list is performed easily with a mouse but not so easily with voice commands, especially when the items are not accessible by the software package. Some mouse commands are simply left out of the current recognition interface. This seems to be outside the realm of what the recognition software needs to do. A less conspicuous problem concerns what the actual commands are called. Sometimes the verbal commands do not have visual counterparts, so the crew must remember the commands they must say rather than looking at a display. The development of these command names is ad hoc and relies on the developer's preferences as to what to name certain commands.

5.4 Generation Strengths

The speech generation provides very comprehensible speech from text strings that are routed from Timeliner. One of the biggest strengths is the fact that the speech is another avenue for feedback to the crew members. With a screen full of displays, it is easy to miss a message displayed to the screen. It turns out to be much more noticeable when the same messages are also spoken. The generation software can change various attributes such as voice pattern, gender, and speed. These provide the means to set up the generation software to suit the crew. We designed the speech generation so that all messages that are displayed on the screen will get spoken in the correct order and so that no messages are lost. This is important for the space station since causality could matter a great deal. It is still unclear whether really old messages should still get spoken. It would be trivial to add a time-of-existence field for messages if deemed necessary.

The interface for the generation is very simple and intuitive. This benefits Timeliner by providing simple access to spoken messages.

5.5 Generation Weaknesses

The generation software has a few areas where it could be improved. In order to get the correct inflection on the voice output, it is necessary to put the text of the message in a sentence-like form rather than sending it word-by-word. This, however, seems reasonable and is usually the way that one would want to organize the output. The most annoying aspect of the generation software is that words that are in all capital letters get spelled. This means that the text either needs to be in mixed case or all lower case to produce sentences. Unfortunately, many of Timeliner's internal commands are capitalized. It would be nice if the software provided a flag that could be set to change this behavior so that it would speak, rather than spell, sentences of all capital letters. However, acronyms get correctly spelled as long as they are in capital letters.

Currently messages are spoken only once. If the crew were busy, it would be desirable for them to be able to request that the message be repeated. This is something that would be possible given better integration between the speech recognition, Timeliner, and the speech generation.

5.6 Open Issues for Generation

An interesting area for research involves associating a priority with the messages. For instance, the crew might want to suppress messages from certain Timeliner scripts and allow messages to be spoken from others. If messages were given different priorities, then messages could reach the crew if they were important enough. Also, messages could signal the crew that they should pay attention, because an important message is about to be spoken. With these abilities, Timeliner would truly start to become an integral participant in station operation, freeing up the crew.

Another issue involves the messages themselves. Currently, the messages spoken are almost identical to the messages displayed to the screen. It might give the script developers more control if there were optional recognition-specific messages that could be associated with every normal message. In this manner, the developers could adapt a message to the specific speaking nuances of the generation software and still have a standard message that gets displayed. Along with this, the generation software could provide more functions if it had more hooks into Timeliner.

5.7 Further Research

There are a lot of topics that still need research in order to better understand the uses of a speech interface with Timeliner. The speech recognition could deal with authentication for example, if the patterns were closely tied to the speaker's individual voice patterns. With this, it would be possible to load up different sets of voice commands for different crew members. This would provide safer operation for the station since each crew member would only have access to the commands they were suppose to use. A less stringent version of this is possible in the current version, but there is no guarantee that the wrong crew member couldn't start up some other crew member's command templates.

Certain commands have serious consequences in a real space mission. At present, it would be unadvisable to allow voice versions of these commands since there is the outside chance that the recognition will make an incorrect match. However, it is possible to add some redundancy checks to the commands so that it would take more than one voice command to initiate certain actions. Or, it is possible to load up different sets of commands based on their criticality.

5.8 Extensions

The next step for this software seems to be a real-time version. This would further verify the utility of a speech interface with Timeliner and prove that it is feasible for on board the International Space Station. Also, further study should be done on the inclusion of a speech interface into the Timeliner baseline. More utility could be gained by further integrating the packages.

An exciting addition to the speech interface would be a small display that travels with the crew--such as a heads-up display. With this, the crew would be completely free, with nothing requiring hand inputs, thereby allowing the crew to better accomplish the experiments. Everything that the crew needed would be headworn.

5.9 Utility

This research was to be a proof-of-concept endeavor. It has shown the utility of a speech interface with Timeliner. The speech recognition complements the current X-window interface by freeing the crew from mouse constraints. In addition, the speech generation provides functionality through extra feedback to the crew with messages from Timeliner. This research shows that it is both economically and technically feasible to include a relatively good speech interface with Timeliner. Hopefully, it will spawn further study, resulting in a working version out in space.

Acknowledgments

I am grateful to all the members of the Timeliner Project at Draper Laboratory. Their hard work and their patience were invaluable. I would especially like to thank Richard Berthold and the Lab for allowing me to undertake this research as a Draper Fellow. I would also like to acknowledge the help I received from Frank Kreimendahl, Tony Bogner, Pete Matthews, John Martin, and Kevin Lee while completing this research.

I appreciate the time that my advisor, Stephanie Seneff put forth in helping me finish my thesis. She was always available when I needed her.

Finally, I would like to thank my family for all their support throughout my school experience. I am at this point in my life due to their unwavering belief in me.

Thanks be to my Lord and Saviour Jesus Christ.

Jim Napier

Appendix A

Timeliner Windows

A.1 Main Window

The following figures show the current Timeliner interface. Each button and scroll bar in the displays have corresponding voice commands to control them.

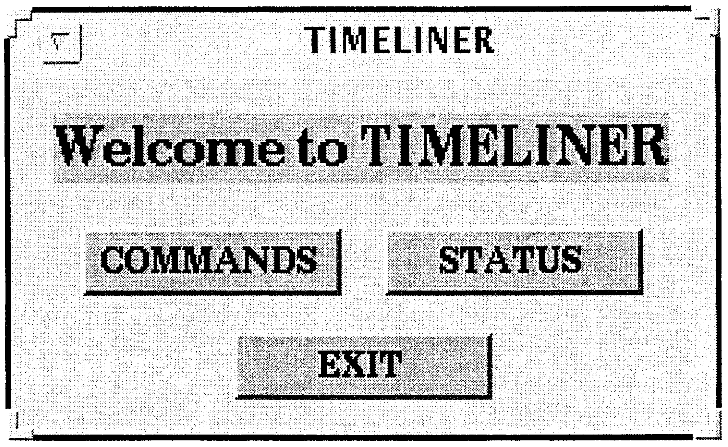
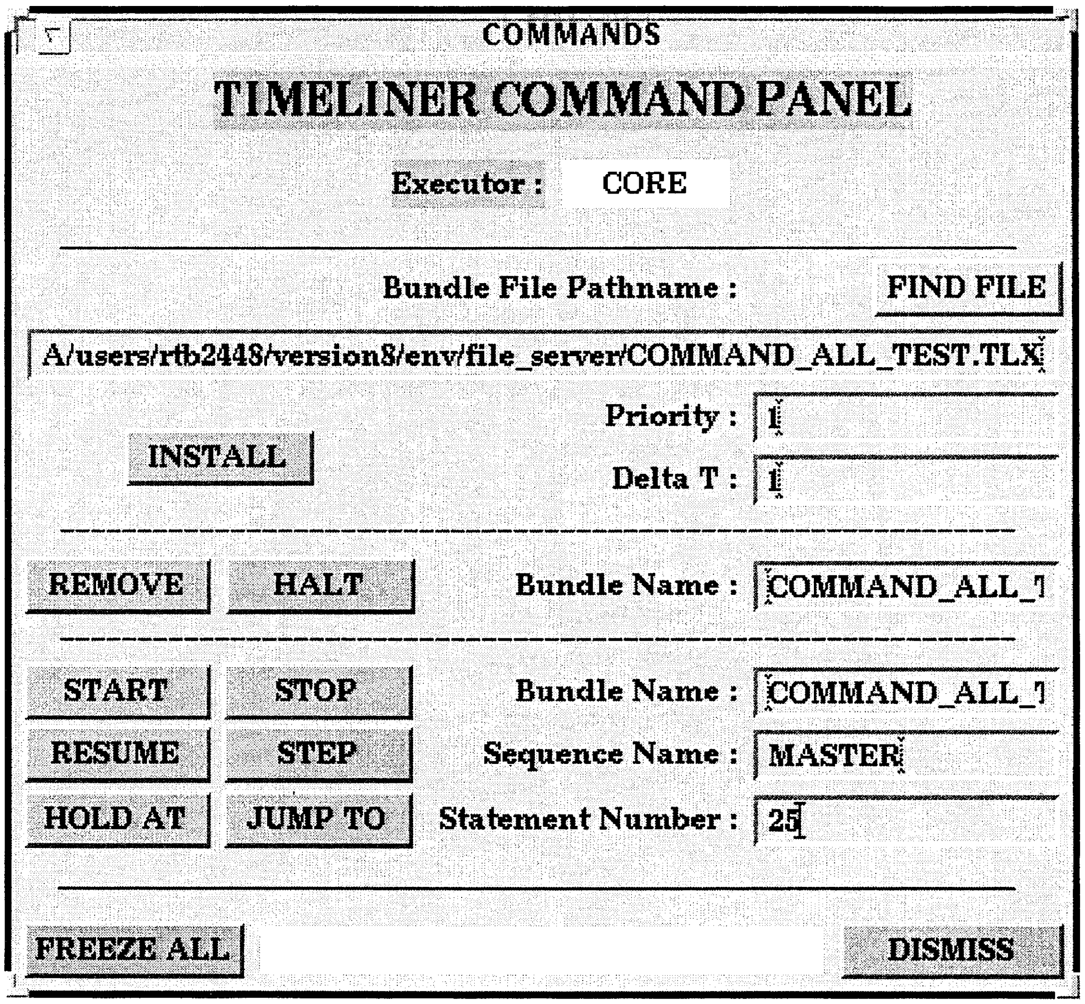


Figure A.1: Timeliner Main Window

Figure A.1 shows the window that initially comes up when Timeliner is started. From this window, the user can open the commands window (to load and install a script) or open the status windows (to view scripts running). This is the only place where the user can exit Timeliner.

A.2 Command Window



Executor : CORE

Bundle File Pathname :

A/users/rtb2448/version8/env/file_server/COMMAND_ALL_TEST.TLX

Priority : 1

Delta T : 1

Bundle Name : COMMAND_ALL_T

Bundle Name : COMMAND_ALL_T

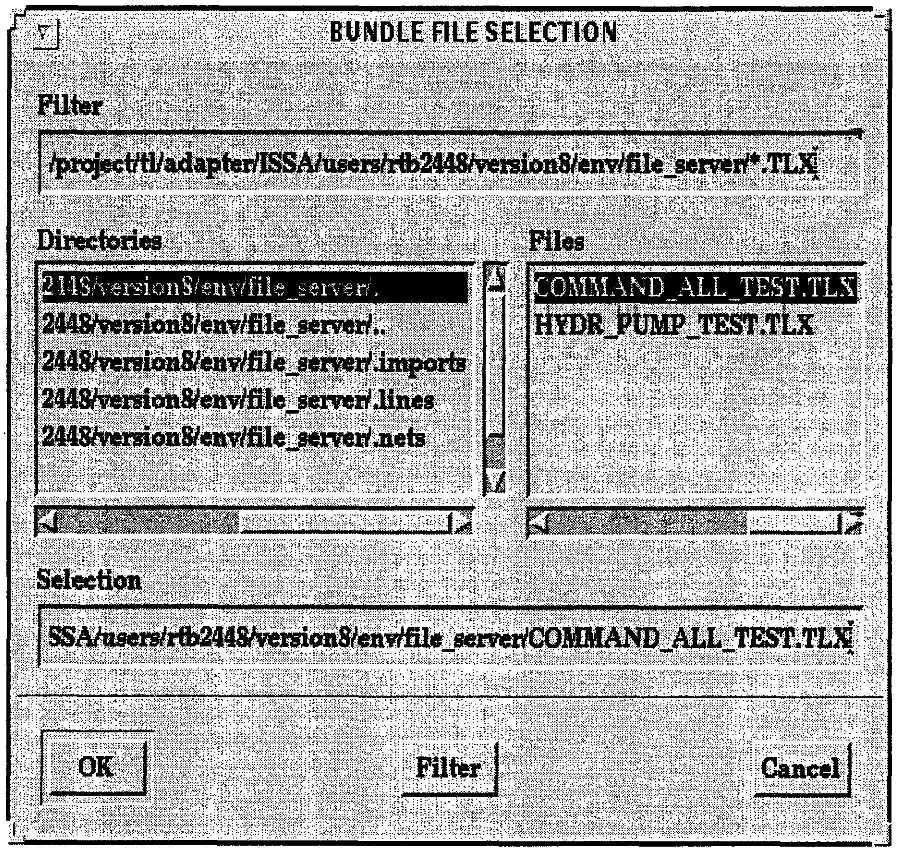
Sequence Name : MASTER

Statement Number : 25

Figure A.2: Commands Window

Figure A.2 shows the commands window. From this window, the user can control bundle installation and a sequence within a bundle. The user can also load up a script from the file server.

A.3 Bundle File Selection Window



/project/tl/adapter/ISSA/users/rfb2448/version8/env/file_server/*.TLX

```
2448/version8/env/file_server/.
2448/version8/env/file_server/..
2448/version8/env/file_server/.imports
2448/version8/env/file_server/.lines
2448/version8/env/file_server/.nets
```

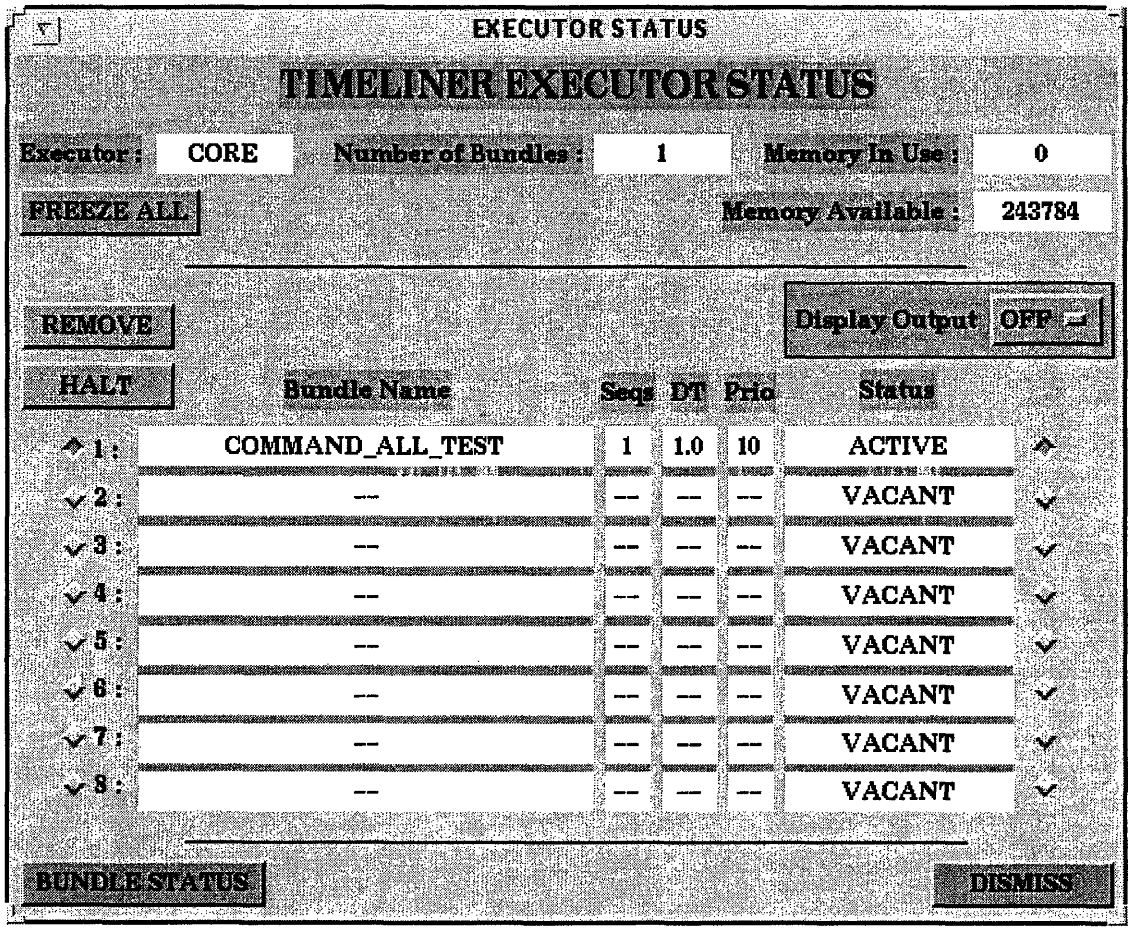
```
COMMAND_ALL_TEST.TLX
HYDR_PUMP_TEST.TLX
```

SSA/users/rfb2448/version8/env/file_server/COMMAND_ALL_TEST.TLX

Figure A.3: Bundle File Selection Window

Figure A.3 shows the bundle file selection window. From this window, the user can search through the file system to find Timeliner scripts to load.

A.4 Executor Status Window



Executor : CORE Number of Bundles : 1 Memory In Use : 0

Memory Available : 243784

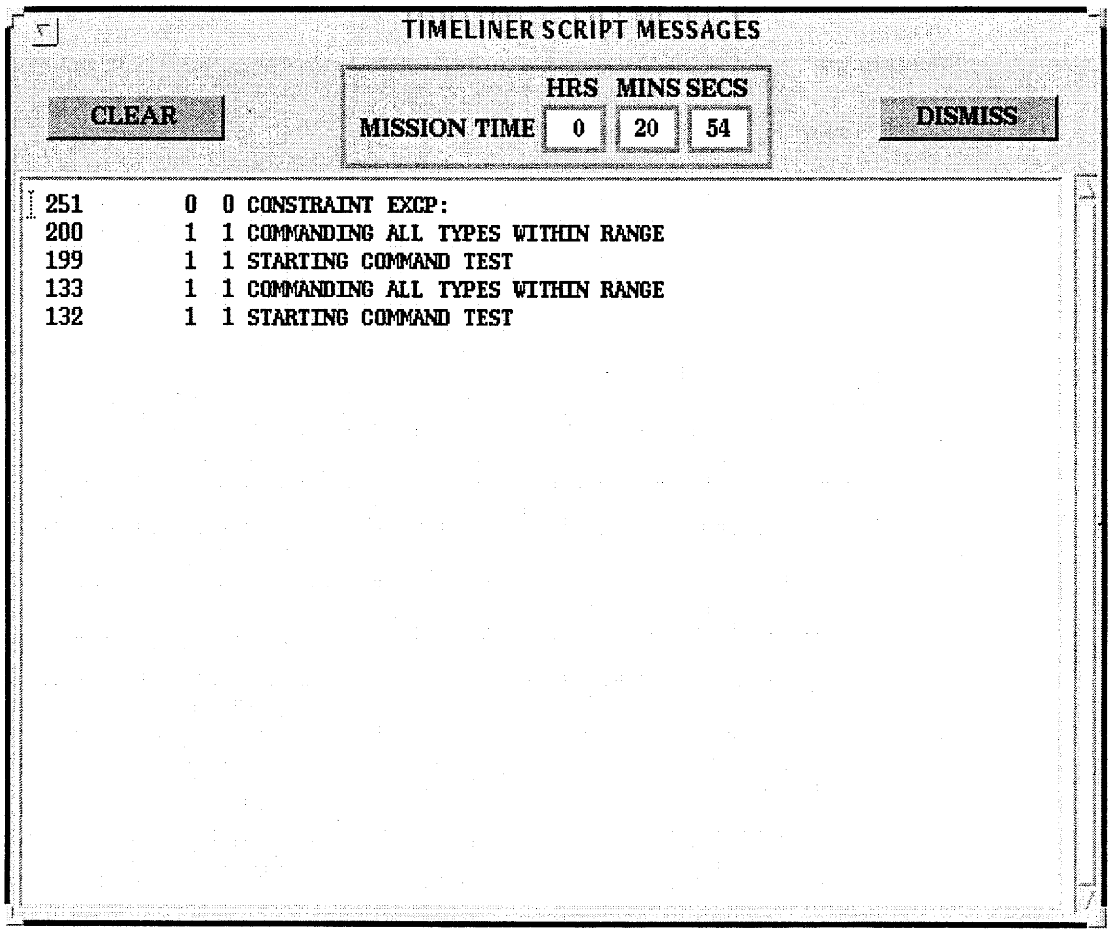
Display Output: OFF

	Bundle Name	Seqs	DT	Prio	Status
1 :	COMMAND_ALL_TEST	1	1.0	10	ACTIVE
2 :	--	--	--	--	VACANT
3 :	--	--	--	--	VACANT
4 :	--	--	--	--	VACANT
5 :	--	--	--	--	VACANT
6 :	--	--	--	--	VACANT
7 :	--	--	--	--	VACANT
8 :	--	--	--	--	VACANT

Figure A.4: Executor Status Window

Figure A.4 shows the executor status window. The user gets information about the bundles which are loaded and can control the loaded bundles. The user can get more information about a particular bundle by pressing the bundle status button.

A.5 Script Messages Window



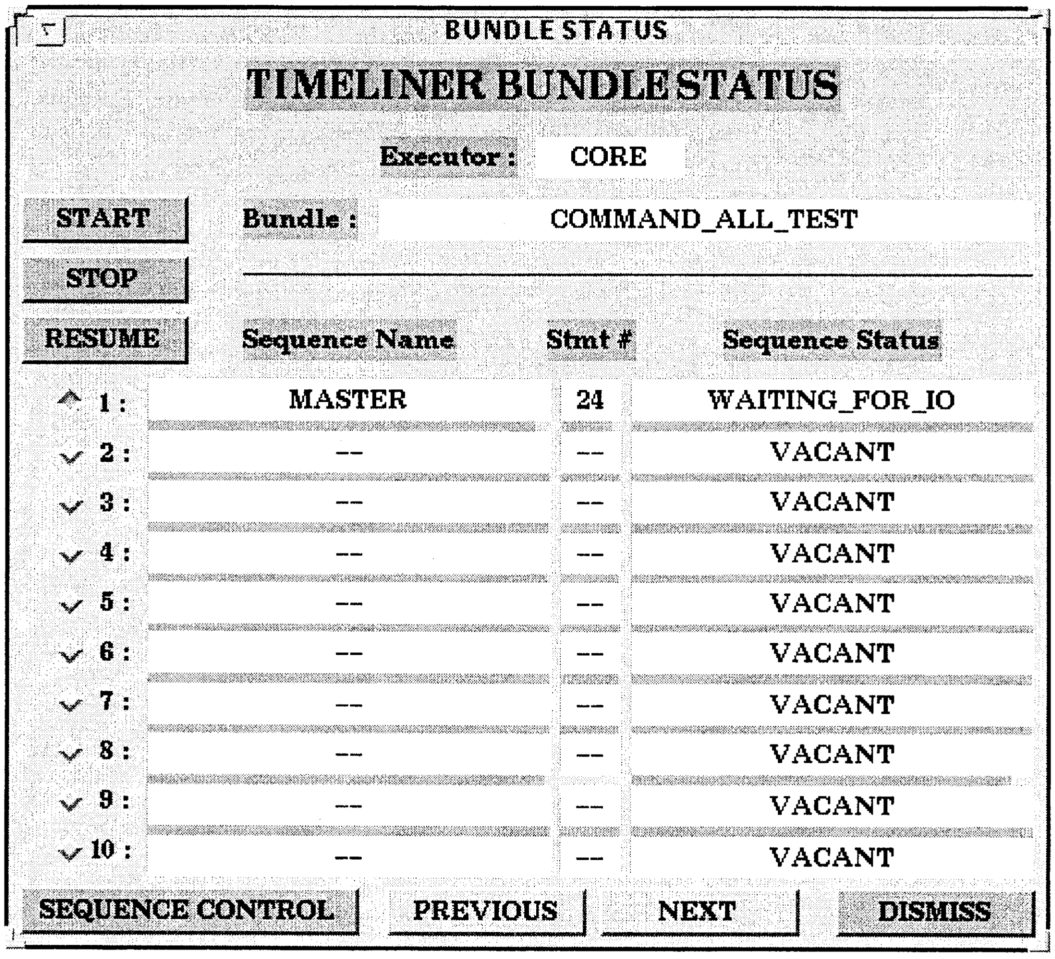
MISSION TIME HRS 0 MINS 20 SECS 54

```
251      0  0 CONSTRAINT EXCP:
200      1  1 COMMANDING ALL TYPES WITHIN RANGE
199      1  1 STARTING COMMAND TEST
133      1  1 COMMANDING ALL TYPES WITHIN RANGE
132      1  1 STARTING COMMAND TEST
```

Figure A.5: Timeliner Script Messages Window

Figure A.5 shows the Timeliner script messages window. The user sees messages displayed from the script being run and from Timeliner's executor. Each message here is spoken using the speech generation software, except the initial numbers for statement number, bundle number, and sequence number.

A.6 Bundle Status Window



Executor : CORE

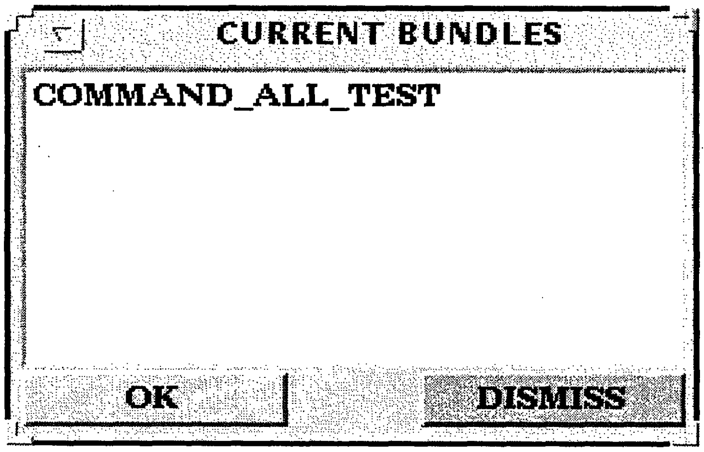
Bundle : COMMAND_ALL_TEST

	Sequence Name	Stmt #	Sequence Status
1 :	MASTER	24	WAITING_FOR_IO
2 :	--	--	VACANT
3 :	--	--	VACANT
4 :	--	--	VACANT
5 :	--	--	VACANT
6 :	--	--	VACANT
7 :	--	--	VACANT
8 :	--	--	VACANT
9 :	--	--	VACANT
10 :	--	--	VACANT

Figure A.6: Bundle Status Window

Figure A.6 shows the bundle status window. The user gets information about a particular bundle, specifically the sequences contained within the bundle. The user can control each sequence within the bundle and can get more information about a particular sequence by pressing the sequence control button.

A.7 Current Bundles Window

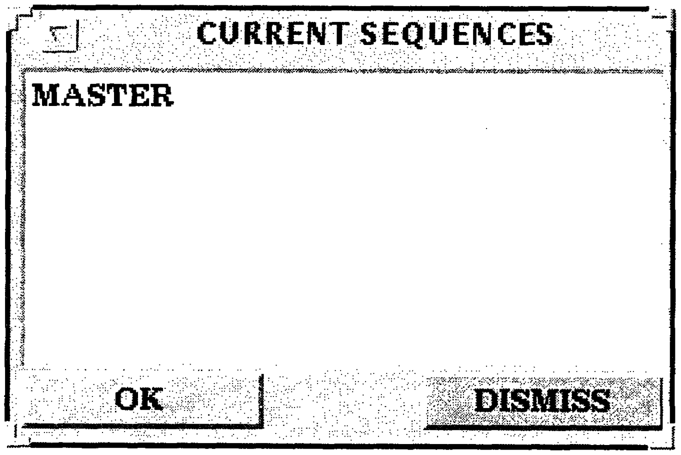


COMMAND_ALL_TEST

Figure A.7: Current Bundles Window

Figure A.7 shows the current bundles window. The user can change what bundle is displayed in the bundle status windows by selecting a bundle in this list.

A.8 Current Sequences Window



MASTER

Figure A.8: Current Sequences Window

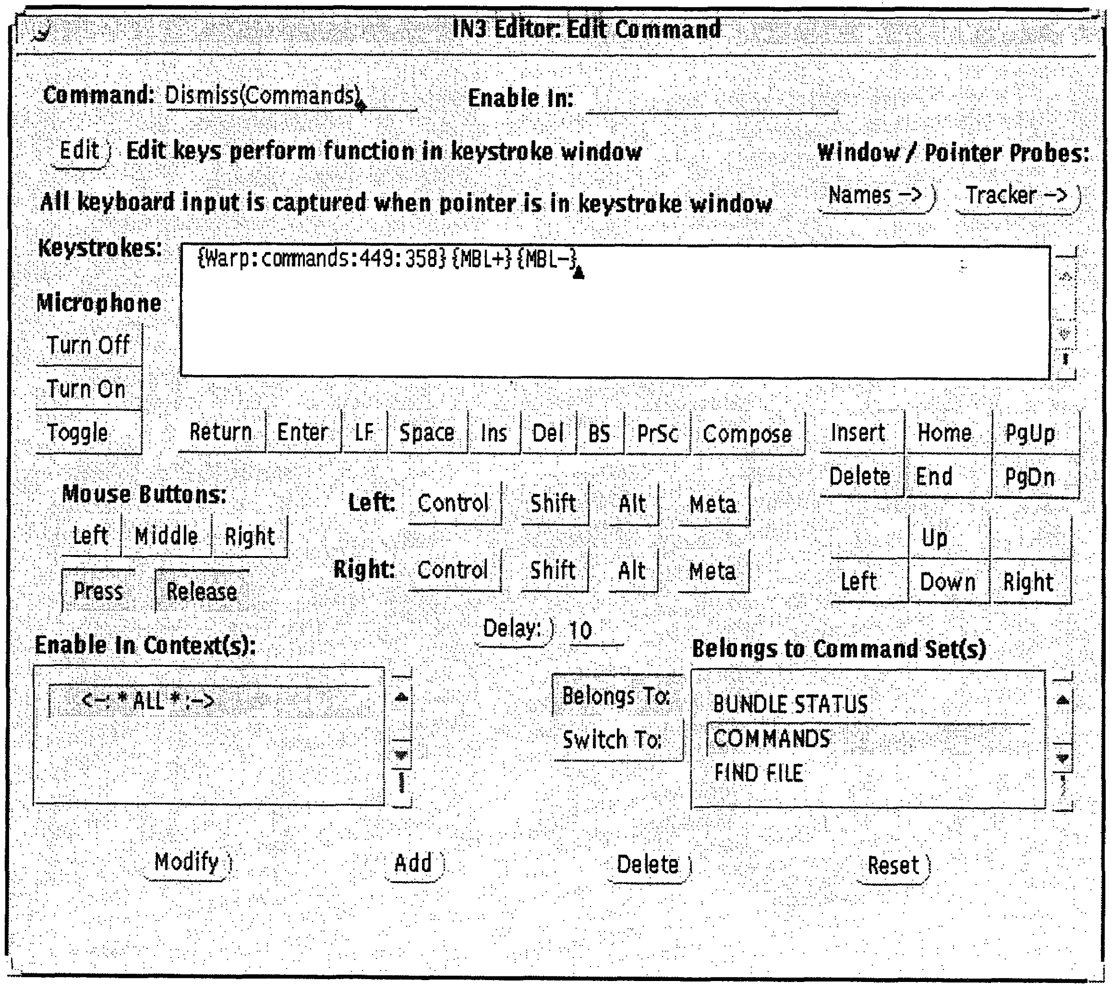
Figure A.8 shows the current sequences window. The user can change what sequence is displayed in the sequence control window by selecting a sequence in this list.

Appendix B

Verbal Commands

B.1 IN^3 Command Window

The following figure represents the editing window for creating verbal commands.



IN3 Editor: Edit Command

Command: Dismiss(Commands) Enable In:

Edit keys perform function in keystroke window

All keyboard input is captured when pointer is in keystroke window

Keystrokes:

```
{Warp:commands:449:358}{MBL+}{MBL-}
```

Delay: 10

Enable In Context(s): <-:*ALL*:->

Belongs to Command Set(s): BUNDLE STATUS, COMMANDS, FIND FILE

Figure B.1: IN^3 Command Edit Window

Figure B.1 shows the edit window in IN^3 for creating new commands or editing existing commands. The edit field is where the user can specify mouse events through the use of the tracker button. Also, command sets to which the commands belong may be specified as well as a transition to a new command set once the command is finished.

B.2 Commands

The following table describes each command; it lists the command set and the action of the command.

Table B.1: List of Commands

Command Name	Command Set	Action
_MICROPHONE	Never Off	Toggle Recognition
Bundle File Selection Window	Find File, Never Off, Timeliner	Focus the Window
Bundle Status	Executor Status	Open Bundle Status Window
Bundle Status Window	Bundle Status, Never Off, Timeliner	Focus the Window
Cancel	Find File	Depress Cancel Button
Clear	Timeliner Script Messages	Depress Clear Button
Commands	Timeliner	Depress Commands Button
Commands Window	Commands, Never Off, Timeliner	Focus the Window
Current Bundles Window	Current Bundles, Never Off, Timeliner	Focus the Window
Current Sequences Window	Current Sequences, Never Off, Timeliner	Focus the Window
Default	Commands	Enter Default Text
Dismiss(Bundle Status)	Bundle Status	Depress Dismiss Button
Dismiss(Commands)	Commands	Depress Dismiss Button
Dismiss(Current Bundles)	Current Bundles	Depress Dismiss Button
Dismiss(Current Seqs)	Current Sequences	Depress Dismiss Button
Dismiss(Exec Status)	Executor Status	Depress Dismiss Button

Table B.1: List of Commands

Command Name	Command Set	Action
Dismiss(Seq Control)	Sequence Control	Depress Dismiss Button
Dismiss(TL Script Msgs)	Timeliner Script Messages	Depress Dismiss Button
Display Auto	Sequence Control	Display Auto Menu
Display Manual	Sequence Control	Display Manual Menu
Display Output Off	Executor Status	Display Output Off Menu
Display Output On	Executor Status	Display Output On Menu
Executor Status Window	Executor Status, Never Off, Timeliner	Focus the Window
Exit	Timeliner	Depress Exit Button
Filter	Find File	Depress Filter Button
Find File	Commands	Depress Find File Button
Freeze All(Commands)	Commands	Depress Freeze All Button
Freeze All(Executor Status)	Executor Status	Depress Freeze All Button
Halt(Commands)	Commands	Depress Halt Button
Halt(Executor Status)	Executor Status	Depress Halt Button
Hold At(Commands)	Commands	Depress Hold At Button
Hold At(Seq Control)	Sequence Control	Depress Hold At Button
Install	Commands	Depress Install Button
Jump To(Commands)	Commands	Depress Jump To Button
Jump To(Seq Control)	Sequence Control	Depress Jump To Button
Next	Bundle Status	Depress Next Button
Okay(Current Bundles)	Current Bundles	Depress Okay Button
Okay(Current Seqs)	Current Sequences	Depress Okay Button
Okay(Find File)	Find File	Depress Okay Button
Page Down(Find File)	Find File	Scroll Down a Page

Table B.1: List of Commands

Command Name	Command Set	Action
Page Down(Seq Ctrl)	Sequence Control	Scroll Down a Page
Page Down(TL Scrpt Msgs)	Timeliner Script Messages	Scroll Down a Page
Page Up(Find File)	Find File	Scroll Up a Page
Page Up(Seq Ctrl)	Sequence Control	Scroll Up a Page
Page Up(TL Scrpt Msgs)	Timeliner Script Messages	Scroll Up a Page
Pick Bundle(Bundle Status)	Bundle Status)	Open Current Bundles Window
Pick Bundle(Seq Ctrl)	Sequence Control	Open Current Bundles Window
Pick Directory	Find File	Select Top Directory
Pick Eight(Bundle Status)	Bundle Status	Pick Eighth Bundle
Pick Eight(Status)	Executor Status	Pick Eighth in List
Pick File	Find File	Select Top File
Pick Five(Bundle Status)	Bundle Status	Pick Fifth Bundle
Pick Five(Status)	Executor Status	Pick Fifth in List
Pick Four(Bundle Status)	Bundle Status	Pick Fourth Bundle
Pick Four(Status)	Executor Status	Pick Fourth in List
Pick Nine(Bundle Status)	Bundle Status	Pick Ninth Bundle
Pick One(Bundle Status)	Bundle Status	Pick First Bundle
Pick One(Status)	Executor Status	Pick First in List
Pick Sequence(Seq Ctrl)	Sequence Control	Open Current Sequences Window
Pick Seven(Bundle Status)	Bundle Status	Pick Seventh Bundle
Pick Seven(Status)	Executor Status	Pick Seventh in List
Pick Six(Bundle Status)	Bundle Status	Pick Sixth Bundle
Pick Six(Status)	Executor Status	Pick Sixth in List
Pick Ten(Bundle Status)	Bundle Status	Pick Tenth Bundle

Table B.1: List of Commands

Command Name	Command Set	Action
Pick Three(Bundle Status)	Bundle Status	Pick Third Bundle
Pick Three(Status)	Executor Status	Pick Third in List
Pick Two(Bundle Status)	Bundle Status	Pick Second Bundle
Pick Two(Status)	Executor Status	Pick Second in List
Previous	Bundle Status	Depress Previous Button
Remove(Commands)	Commands	Depress Remove Button
Remove(Executor Status)	Executor Status	Depress Remove Button
Resume(Bundle Status)	Bundle Status	Depress Resume Button
Resume(Commands)	Commands	Depress Resume Button
Resume(Seq Control)	Sequence Control	Depress Resume Button
Scroll Down(Find File)	Find File	Scroll Down
Scroll Down(Seq Ctrl)	Sequence Control	Scroll Down
Scroll Down(TL Scrpt Msgs)	Timeliner Script Messages	Scroll Down
Scroll Left(Find File)	Find File	Scroll Left
Scroll Left(Seq Ctrl)	Sequence Control	Scroll Left
Scroll Right(Find File)	Find File	Scroll Right
Scroll Right(Seq Ctrl)	Sequence Control	Scroll Right
Scroll Up(Find File)	Find File	Scroll Up
Scroll Up(Seq Ctrl)	Sequence Control	Scroll Up
Scroll Up(TL Scrpt Msgs)	Timeliner Script Messages	Scroll Up
Select Next	Current Bundles, Current Sequences, Find File, Sequence Control	Select Next in List
Select Previous	Current Bundles, Current Sequences, Find File, Sequence Control	Select Previous in List
Sequence Control	Bundle Status	Depress Sequence Control Button

Table B.1: List of Commands

Command Name	Command Set	Action
Sequence Control Window	Never Off, Sequence Control, Timeliner	Focus the Window
Start(Bundle Status)	Bundle Status	Depress Start Button
Start(Commands)	Commands	Depress Start Button
Start(Seq Control)	Sequence Control	Depress Start Button
Status	Timeliner	Depress Status Button
Step(Commands)	Commands	Depress Step Button
Step(Seq Control)	Sequence Control	Depress Step Button
Stop(Bundle Status)	Bundle Status	Depress Stop Button
Stop(Commands)	Commands	Depress Stop Button
Stop(Seq Control)	Sequence Control	Depress Stop Button
Timeliner Script Messages Window	Never Off, Timeliner, Timeliner Script Messages	Focus the Window
Timeliner Window	Never Off, Timeliner	Focus the Window
Update	Sequence Control	Depress Update Button

Appendix C

Program Code

C.1 Ada Code for Speech Generation

The following code is designed to be called by a Timeliner procedure with a message to be spoken. This code, in turn, calls some C code to spawn the process of speaking the message via a Tcl/TK script.

Table C.1: TL_Speech_Gen_Interface Specification

```
package TL_speech_gen_interface is
 procedure speak_TL_message(the_msg : in STRING);
end TL_speech_gen_interface;
```

Table C.2: TL_Speech_Gen_Interface Body

```
with SYSTEM;
with TEXT_IO;
with LANGUAGE;

package body TL_speech_gen_interface is

use LANGUAGE;

  --C Routine to send text to speech generation module

  procedure speak_message(the_msg : in SYSTEM.ADDRESS);

  pragma INTERFACE(C, speak_message);
  pragma INTERFACE_NAME(speak_message, C_SUBP_PREFIX &
  "speak_message");
  pragma LINK_WITH("speak_message.o");


  --Procedure to handle formatting msg correctly
  procedure speak_TL_message(the_msg : in STRING) is

    msg_length : constant NATURAL := the_msg'LENGTH;
    C_msg_length: constant NATURAL := NATURAL'SUCC(msg_length);
    C_msg:STRING(1..C_msg_length);

  begin

    --convert ada string into C string
    C_msg(1..msg_length) := the_msg;
    C_msg(C_msg_length) := ASCII.NUL;

    --call the c routine to speak the message
    speak_message(C_msg(1)'ADDRESS);

    --catch all exceptions because we don't want this to kill Timeliner
    exception
      when others =>
        text_io.put_line("Couldn't output audible message.");

  end speak_TL_message;

end TL_speech_gen_interface;
```

C.3 C code for Speech Generation

The following code calls the TCL scripts that are responsible for calling the speech generation software.

Table C.3: Speak_Message C Procedure

```
#include <stdio.h>
#include <stdlib.h>

#define MAXSTRING 100

void speak_message(char *the_msg)
{

  char command[MAXSTRING], *tmp_file_name;
  FILE *ifp;

/*need a variable to know when we've called this proc more
  than once*/
  static int more_than_once = 0;

  tmp_file_name = tmpnam(NULL);
/*  tmp_file_name = "TLmsg_str.txt";*/
  if (!more_than_once) {
    sprintf(command, "wish -f ttsayonce.tcl %s",
          tmp_file_name);
    more_than_once++;}
  else {
    sprintf(command, "wish -f ttsayagain.tcl %s",
          tmp_file_name);}
  ifp = fopen(tmp_file_name, "w");
  fprintf(ifp, "%s", the_msg);
  fclose(ifp);

  system(command);

  remove(tmp_file_name);

}
```

C.4 Tcl/TK Scripts

The following code is the two scripts necessary to enable speech generation. They call the speech generation software through the Tcl command TTplay.

Table C.4: Tcl Script TTSayOnce.Tcl

```
# Read text, either from the file if a name was given,
# or from stdin if no arguments were given,
# and send to TrueTalk
#
# Run this file (ttsayonce.tcl) by entering:
#   wish -f ttsayonce.tcl
# or
#   wish -f ttsayonce.tcl filename

wm withdraw .

if { $argc > 1 } {
   puts "Usage: $argv0 \[filename\]"
   exit 1
}

if { $argc == 1 } {
   set infile [open [lindex $argv 0] r]
#   puts "Sending contents of [lindex $argv 0] to TrueTalk."
} else {
   set infile stdin
   puts "Reading input from stdin."
}

set text [read $infile]

#check to make sure truetalk is running
set x [lsearch [winfo interps] ttgui.tcl]
if {$x == -1} {
  # it isn't running so don't send the text to truetalk
  puts "TrueTalk is not running, so the messages will not be spoken."
} else {

  puts "About to speak->$text<-"

  catch {send ttgui.tcl after 0 TTplay \{\{$text\}\}  } m
}

exit 0
```

Table C.5: Tcl Script TTSayAgain.Tcl

```
wm withdraw .

if { $argc > 1 } {
   puts "Usage: $argv0 \[filename\]"
   exit 1
}

if { $argc == 1 } {
   set infile [open [lindex $argv 0] r]
#   puts "Sending contents of [lindex $argv 0] to TrueTalk."
} else {
   set infile stdin
   puts "Reading input from stdin."
}

set text [read $infile]

#check to make sure truetalk is running
set x [lsearch [winfo interps] ttgui.tcl]
if {$x == -1} {
  # it isn't running so don't send the text to truetalk
  puts "TrueTalk is not running, so the messages will not be spoken."
} else {


  #The variable TTplaying must already exist
  #In order to insure that TTplaying already exists
  #a call to TTplay must take place before this program is called.

  set vrb [send ttgui.tcl {set tmpr $TTplaying}]
  if {$vrb == 1} {
     puts "Waiting to speak->$text<-"}

  while {$vrb == 1} {
     set vrb [send ttgui.tcl {set tmpr $TTplaying}]}

  puts "About to speak->$text<-"

  catch {send ttgui.tcl after 0 TTplay \{\{$text\}\}  } m
}

exit 0
```

Appendix D

Timeliner Script

D.1 Demonstration Script

The following code is a Timeliner script designed specifically to show the usefulness of a speech interface for Timeliner. It was created from real data obtained from the Marshall Space Flight Center in Huntsville, Alabama. It comes from a payload experiment.

The AGHF is a Bridgeman furnace designed for directional solidification of semiconductor crystals. The AGHF consists of three modules: the electronics module, the core facility module, and the gas storage module. The furnace chamber pressure is controlled by a sensor, and goes below the Spacelab pressure of 1013 bar, but during sample extraction, a slight overpressure of 7 mbar will aid the astronaut.

```
--This Timeliner script is derived from a payload operating procedure designed
--to run an AGHF Furnace experiment.

BUNDLE AGHF_Experiment

 DECLARE In_Time BOOLEAN
 DECLARE Start_Time NUMERIC
 DECLARE Stop_Time NUMERIC

 --The following variables would really be external for the experiment
 --but are internal here just for ease of development and testing
 DECLARE Exp_Vent_Bleed_Vlv BOOLEAN
 DECLARE Exp_Vent_Vlv  BOOLEAN
 DECLARE Argon1_2_Main  BOOLEAN
 DECLARE Process_Finished_Light  BOOLEAN
 DECLARE Hot_Sample_Light  BOOLEAN
 DECLARE Setup_Sample_Light  BOOLEAN
 DECLARE Reprogramming_Light  BOOLEAN
 DECLARE Preheating_Phase_Light  BOOLEAN
DECLARE Vacuum_Valves_Open_Light  BOOLEAN
DECLARE Operations_Light  BOOLEAN
DECLARE Failure_Light  BOOLEAN
DECLARE EP_Off_Light  BOOLEAN
DECLARE EP_On_Light  BOOLEAN
DECLARE Chamber_Open_Light  BOOLEAN
DECLARE Load_PROM_Light  BOOLEAN
DECLARE Power_Light  BOOLEAN

--Definitions for variables
DEFINE OPEN AS TRUE
DEFINE CLOSED AS FALSE



-----------------
--This is the main sequence for controlling the experiment
-----------------

SEQUENCE Master ACTIVE

  --needed to allow for the internal variables to ease development
  CALL Initialize_Variables

  WAIT 5.0
  MESSAGE "This is a script designed to control a furnace experiment in space."
  WAIT 5.0

  --activate it
  CALL Setup_Activation

  --Done with Setup and Activation so simulate the actual physical process
  CALL Automatic_Run

  --deactivate it
  CALL Deactivation_Shutdown

  --remove cartridges
  --not included to speed up run
  --    CALL Cartridge_Removal
```

```
    WAIT 5.0
    MESSAGE "Done with the experiment.  Have a nice day."

  CLOSE SEQUENCE Master




  SUBSEQUENCE Setup_Activation

    MESSAGE "Flip the Lamp switch on."
    WAIT 1.0
    MESSAGE "Resume to continue." PAUSE

    MESSAGE "Check that the lamps are operating."
    WAIT 2.0
    MESSAGE "If lamps not operating then notify POCC."
    WAIT 3.0
    MESSAGE "Resume to continue." PAUSE

    --Start automatic checks for malfunctions
    CALL Start_Malfunction_Procedures

    MESSAGE "Note:  Process chamber lamps will remain on throughout the experi-
ment."
    WAIT 2.0

    MESSAGE "Check for obvious damage inside chamber through viewport, such as"
    MESSAGE " condensation on mirror, glass particles, damage to LMR locking fin-
gers."
    MESSAGE "Resume to continue." PAUSE

    IF ((Exp_vent_bleed_vlv /= CLOSED) or(Exp_vent_vlv /= OPEN)) THEN
      MESSAGE "Notify POCC.  Problem with vent or relief valve."
      WAIT 1.0
      MESSAGE "Resume to continue." PAUSE
    END IF

    MESSAGE "Check that the manual valve is open, otherwise notify POCC."
    WAIT 1.0
```

```
MESSAGE "Resume to continue." PAUSE

MESSAGE "Unstow gray tape and temp stow."
WAIT 1.0
MESSAGE "Resume to continue." PAUSE

MESSAGE "Unstow experiment plug, tether to EGSE J72 connector cap."
WAIT 1.0
MESSAGE "Resume to continue." PAUSE

MESSAGE "Turn lock handle to Unlocked Position after releasing it by"
MESSAGE " pushing the manual latch outside while manually pushing the"
MESSAGE " release bolt."
WAIT 1.0
MESSAGE "Resume to continue." PAUSE

MESSAGE "Turn the diffuser locking nut of locking cartridge"
MESSAGE " counter-clockwise about 7 revolutions."
WAIT 3.0
MESSAGE "Push the diffuser locking nut to loosen center rod of locking cartridge."
WAIT 3.0
MESSAGE "Check that the center rod is moving freely in and out."
WAIT 1.0
MESSAGE "Resume to continue." PAUSE

MESSAGE "Caution.  After 7 counter-clockwise revolutions of the locking"
MESSAGE " wheel, the locking cartridge becomes free."
MESSAGE "Contact of the locking cartridge with the interior of the"
MESSAGE " chamber and time of open chamber should be minimized."
WAIT 4.0

MESSAGE "Turn locking wheel of locking cartridge counter-clockwise 7"
MESSAGE " revolutions and remove locking cartridge."
WAIT 1.0
MESSAGE "Resume to continue." PAUSE

MESSAGE "Insert experiment plug to close chamber port."
WAIT 1.0
MESSAGE "Resume to continue." PAUSE
```

```
	MESSAGE "Turn lock handle counter-clockwise to Locked Position."
	WAIT 1.0
	MESSAGE "Resume to continue." PAUSE

	MESSAGE "Temp stow locking cartridge with gray tape."
	WAIT 1.0
	MESSAGE "Resume to continue." PAUSE

	MESSAGE "Notify POCC, Record temp stow location."
	WAIT 1.0
	MESSAGE "Resume to continue." PAUSE

	MESSAGE "Unstow Sample Simulation connector and connect to J71 connector
saver."
	WAIT 1.0
	MESSAGE "Resume to continue." PAUSE

	MESSAGE "Flip the Power switch on."
	WAIT 1.0
	MESSAGE "Resume to continue." PAUSE

	WAIT 10.0 --really wait 2 minutes

	MESSAGE "Check that the EP Off light is off, the Failure light is off,"
	MESSAGE " and that all other lights are on."
	WAIT 1.0
	MESSAGE "Resume to continue." PAUSE

	MESSAGE "Check with POCC (to verify telemetry data transmission)."
	WAIT 1.0
	MESSAGE "Resume to continue." PAUSE

	--no commands to ease speed of testing
	--   COMMAND Enable_EMONS ...

	MESSAGE "Check that all AGHF parameters are enabled."
	WAIT 1.0
	MESSAGE "Resume to continue." PAUSE

	SET Argon1_2_main TO OPEN
 MESSAGE "Stow the gray tape."
 WAIT 1.0
 MESSAGE "Resume to continue." PAUSE

CLOSE SUBSEQUENCE Setup_Activation




SUBSEQUENCE Deactivation_Shutdown

 --no commands to ease speed of testing
 --    COMMAND Inhibit_EMONS

 MESSAGE "Check that all AGHF parameters are inhibited"
 WAIT 1.0
 MESSAGE "Resume to continue." PAUSE

 IF (Process_Finished_Light = OFF) THEN
  MESSAGE "Notify POCC that the Process Finished light is off."
  WAIT 1.0
  MESSAGE "Resume to continue." PAUSE
 END IF

 MESSAGE "Notify POCC if the lamps are not operating."
 WAIT 1.0
 MESSAGE "Resume to continue." PAUSE

 MESSAGE "Check for obvious damage inside chamber."
 WAIT 1.0
 MESSAGE "Resume to continue." PAUSE

 MESSAGE "Flip the Start/Continue switch up."
 WAIT 1.0
 MESSAGE "Resume to continue." PAUSE

 IF (Hot_Sample_Light = ON) THEN
  MESSAGE "The sample is still too hot."
 END IF
```

```
MESSAGE "Check that the Hot Sample light is off."
WAIT 1.0
MESSAGE "Resume to continue." PAUSE

MESSAGE "Waiting for two minutes."
WAIT 10.0 --really should be 120.0

IF (Setup_Sample_Light = OFF) THEN
 MESSAGE "Notify POCC that the Set Up Sample light is off."
 WAIT 1.0
 MESSAGE "Resume to continue." PAUSE
END IF

MESSAGE "Unstow the locking cartridge."
WAIT 1.0
MESSAGE "Resume to continue." PAUSE

MESSAGE "The following directions must be completed within 40 seconds."

SET In_Time TO FALSE
EVERY 1.0 BEFORE In_Time THEN
 SET Start_Time TO TIME
 MESSAGE "Press the Press-To-Release-Lock Handle up."
 WAIT 1.0
 MESSAGE "Resume to continue." PAUSE
 MESSAGE "Check that the Released light is on."
 WAIT 1.0
 MESSAGE "Resume to continue." PAUSE
 MESSAGE "Turn lock handle to Unlocked Position."
 MESSAGE "If the Press-To-Release-Lock Handle doesn't retract,"
 MESSAGE "Finger push it back into place."
 WAIT 1.0
 MESSAGE "Resume to continue." PAUSE
 SET Stop_Time TO TIME
 IF (Stop_Time - Start_Time > 40.0) THEN
--Took too long
   MESSAGE "Too much time has elapsed.  Try again."
 ELSE
   SET In_Time TO TRUE
 END IF
END EVERY

MESSAGE “Mark the cartridge label with a check.”
WAIT 1.0
MESSAGE “Resume to continue.”

MESSAGE “Disconnect cartridge cable from the Pulse Marking Connector”
MESSAGE “ and the Sample Signal Connector.”
WAIT 1.0
MESSAGE “Resume to continue.” PAUSE

MESSAGE “Unstow and don deerskin gloves.”
WAIT 1.0
MESSAGE “Resume to continue.” PAUSE

MESSAGE “Unstow processed cartridge’s protective sheath and temp stow.”
WAIT 1.0
MESSAGE “Resume to continue.” PAUSE

MESSAGE “Damage to the process chamber interior and to the cartridge and”
MESSAGE “cartridge port inner surface should be avoided.  Time of open”
MESSAGE “should be minimized.”
WAIT 1.0
MESSAGE “Warning! Cartridge may be hot.”
WAIT 2.0

MESSAGE “Open velcro strap to release cartridge tether while holding cartridge.”
WAIT 1.0
MESSAGE “Resume to continue.” PAUSE

MESSAGE “Remove processed cartridge from process chamber.”
WAIT 1.0
MESSAGE “Resume to continue.” PAUSE

MESSAGE “Disconnect cartridge cable from cartridge connector.”
WAIT 1.0
MESSAGE “Resume to continue.” PAUSE

MESSAGE “Insert processed cartridge into protective sheath.”
```

```
        WAIT 1.0
        MESSAGE "Resume to continue." PAUSE

        MESSAGE "Stow the processed cartridge, cartridge cable, and simulation cable."
        WAIT 1.0
        MESSAGE "Resume to continue." PAUSE

        MESSAGE "Doff and stow deerskin gloves."
        WAIT 1.0
        MESSAGE "Resume to continue." PAUSE

        MESSAGE "Do not scratch the cartridge port inner surface."
        MESSAGE "By watching through viewport, choose a proper orientation of the"
        MESSAGE " locking cartridge crankpin to avoid damage to the LMR locking fin-
gers"
        MESSAGE " which are on the cooling zone."
        WAIT 2.0

        MESSAGE "Using locking cartridge alignment mark, insert locking cartridge"
        MESSAGE " while observing through viewport."
        WAIT 1.0
        MESSAGE "Resume to continue." PAUSE

        MESSAGE "Turn locking wheel of locking cartridge clockwise 7 revolutions."
        WAIT 1.0
        MESSAGE "Resume to continue." PAUSE

        MESSAGE "Turn lock handle counter-clockwise to Locked Position."
        WAIT 1.0
        MESSAGE "Resume to continue." PAUSE

        MESSAGE "Tightly screw the diffuser locking nut of locking cartridge."
        WAIT 1.0
        MESSAGE "Resume to continue." PAUSE

        MESSAGE "Remove Prom."
        WAIT 1.0
        MESSAGE "Resume to continue." PAUSE

        MESSAGE "Stow Prom."
```

```
   WAIT 1.0
   MESSAGE “Resume to continue.” PAUSE

   MESSAGE “Disconnect and stow experiment plug.”
   WAIT 1.0
   MESSAGE “Resume to continue.” PAUSE

   MESSAGE “Replace three connector covers.”
   WAIT 1.0
   MESSAGE “Resume to continue.” PAUSE

   SET ARGON1_2_Main TO CLOSED

   --Stop automatic malfunction checks
   CALL Stop_Malfunction_Procedures

   MESSAGE “Flip Power switch off.”
   WAIT 1.0
   MESSAGE “Resume to continue.” PAUSE

   MESSAGE “Flip Lamp switch off.”
   WAIT 1.0
   MESSAGE “Resume to continue.” PAUSE


  CLOSE SUBSEQUENCE Deactivation_Shutdown




  SUBSEQUENCE Cartridge_Removal

   --Should really be done for each cartridge

   MESSAGE “Check with POCC that mandatory sample cooldown time has
expired.”
   WAIT 1.0
   MESSAGE “Resume to continue.” PAUSE

   MESSAGE “Check that the Process Finished light is on.”
```

```
      WAIT 1.0
      MESSAGE "Resume to continue." PAUSE

      MESSAGE "Notify POCC if lamps are not illuminated."
      WAIT 1.0
      MESSAGE "Resume to continue." PAUSE

      MESSAGE "Check for obvious damage inside chamber through viewport such as"
      MESSAGE " condensation on mirror, glass particles, damage to LMR locking fin-
gers."
      WAIT 1.0
      MESSAGE "Resume to continue." PAUSE

      MESSAGE "Flip Start/Continue switch up firmly and quickly."
      MESSAGE "Immediate response may not occur."
      WAIT 1.0
      MESSAGE "Resume to continue." PAUSE

      MESSAGE "Check that the Hot Sample light is off."
      WAIT 1.0
      MESSAGE "Resume to continue." PAUSE

      WAIT 10.0 --really should wait 2 minutes.
      MESSAGE "Check that the Set Up Sample light is on."
      WAIT 1.0
      MESSAGE "Resume to continue." PAUSE

      SET In_Time TO FALSE
      EVERY 1.0 BEFORE In_Time THEN
        SET Start_Time TO TIME
        MESSAGE "Press the Press-To-Release-Lock Handle up."
        WAIT 1.0
        MESSAGE "Resume to continue." PAUSE
        MESSAGE "Check that the Released light is on."
        WAIT 1.0
        MESSAGE "Resume to continue." PAUSE
        MESSAGE "Turn lock handle to Unlocked Position."
        MESSAGE "It the Press-To-Release-Lock Handle doesn't retract,"
        MESSAGE "Finger push it back into place."
        WAIT 1.0
```

```
    MESSAGE "Resume to continue." PAUSE
    SET Stop_Time TO TIME
    IF (Stop_Time - Start_Time > 40.0) THEN
--Took too long
      MESSAGE "Too much time has elapsed.  Try again."
    ELSE
      SET In_Time TO TRUE
    END IF
  END EVERY

  MESSAGE "Unstow wipes and temp stow."
  WAIT 1.0
  MESSAGE "Resume to continue." PAUSE

  MESSAGE "Mark the cartridge label with a check."
  WAIT 1.0
  MESSAGE "Resume to continue." PAUSE

  MESSAGE "Disconnect cartridge cable from the Pulse Marking Connector"
  MESSAGE " and the Sample Signal Connector."
  WAIT 1.0
  MESSAGE "Resume to continue." PAUSE

  MESSAGE "Unstow and don deerskin gloves."
  WAIT 1.0
  MESSAGE "Resume to continue." PAUSE

  MESSAGE "Unstow processed cartridge's protective sheath and temp stow."
  WAIT 1.0
  MESSAGE "Resume to continue." PAUSE

  MESSAGE "Damage to the process chamber interior and to the cartridge and"
  MESSAGE "cartridge port inner surface should be avoided.  Time of open"
  MESSAGE "should be minimized."
  WAIT 1.0
  MESSAGE "Warning! Cartridge may be hot."
  WAIT 2.0

  MESSAGE "Open velcro strap to release cartridge tether while holding cartridge."
  WAIT 1.0
```

```
MESSAGE "Resume to continue." PAUSE

MESSAGE "Remove processed cartridge from process chamber."
WAIT 1.0
MESSAGE "Resume to continue." PAUSE

MESSAGE "Disconnect cartridge cable from cartridge connector."
WAIT 1.0
MESSAGE "Resume to continue." PAUSE

MESSAGE "Insert processed cartridge into protective sheath."
WAIT 1.0
MESSAGE "Resume to continue." PAUSE

MESSAGE "Stow the processed cartridge, and temp stow the cartridge cable."
WAIT 1.0
MESSAGE "Resume to continue." PAUSE

MESSAGE "Doff and temp stow deerskin gloves."
WAIT 1.0
MESSAGE "Resume to continue." PAUSE

MESSAGE "Insert the Experiment Plug."
WAIT 1.0
MESSAGE "Resume to continue." PAUSE

MESSAGE "Turn lock handle counter-clockwise to Locked Position."
WAIT 1.0
MESSAGE "Resume to continue." PAUSE

MESSAGE "Check that the Chamber Open light is off."
WAIT 1.0
MESSAGE "Resume to continue." PAUSE

MESSAGE "Connect Sample Simulation connector to J71 connector saver."
WAIT 1.0
MESSAGE "Resume to continue." PAUSE

--  COMMAND Close_Chamber_Vacuum_Vlv
```

```
MESSAGE “Note: the furnace chamber vacuum valve has now been powered off”
MESSAGE “ and the valve has been closed. The Vacuum Valves Open light “
MESSAGE “ will come on in about 5 steps.”

WAIT 2.0

MESSAGE “Flip the Press-To-Release-Lock Handle up.”
WAIT 1.0
MESSAGE “Resume to continue.” PAUSE

MESSAGE “Flip the Start/Continue switch up.”
WAIT 1.0
MESSAGE “Resume to continue.” PAUSE

MESSAGE “Check that the Load Prom light is on.”
WAIT 1.0
MESSAGE “Resume to continue.” PAUSE

MESSAGE “Verify a Prom is attached (either color).”
WAIT 1.0
MESSAGE “Resume to continue.” PAUSE

MESSAGE “Flip the Start/Continue switch up.”
WAIT 1.0
MESSAGE “Resume to continue.” PAUSE

MESSAGE “Check that the Reprogramming light is on.”
WAIT 1.0
MESSAGE “Resume to continue.” PAUSE

MESSAGE “Flip the Start/Continue switch up.”
WAIT 1.0
MESSAGE “Resume to continue.” PAUSE

MESSAGE “Check that the Preheating Phase light is on.”
WAIT 1.0
MESSAGE “Resume to continue.” PAUSE

MESSAGE “Check that the Furnace Movement Position is decreasing towards 0.0.”
WAIT 1.0
```

```
 MESSAGE "Resume to continue." PAUSE

 MESSAGE "Notify POCC that furnace translation has started."
 WAIT 1.0
 MESSAGE "Resume to continue." PAUSE

CLOSE SUBSEQUENCE Cartridge_Removal


SUBSEQUENCE Automatic_Run

 MESSAGE "Notify POCC before starting automatic run."
 WAIT 1.0
 MESSAGE "Resume to continue." PAUSE

 IF (Reprogramming_Light = OFF) THEN
  MESSAGE "Warning: Notify POCC that the reprogramming light is off."
  WAIT 1.0
  MESSAGE "Resume to continue." PAUSE
 END IF

 MESSAGE "Flip the Chamber Vacuum Valve Switch to Automatic."
 WAIT 1.0
 MESSAGE "Resume to continue." PAUSE

 MESSAGE "Flip the Start/Continue switch up."
 WAIT 1.0
 MESSAGE "Resume to continue." PAUSE

 IF (Preheating_Phase_Light = OFF) THEN
  MESSAGE "Warning: Notify POCC that the Preheating Phase light is off."
  WAIT 1.0
  MESSAGE "Resume to continue." PAUSE
 END IF

 IF (Vacuum_Valves_Open_Light = OFF) THEN
  MESSAGE "Warning: Notify POCC that the Vacuum Valves Open light is off."
  WAIT 1.0
  MESSAGE "Resume to continue." PAUSE
 END IF
```

```
  IF (Operations_Light = OFF) THEN
    MESSAGE "Warning: Notify POCC that the Operation light is off."
    WAIT 1.0
    MESSAGE "Resume to continue." PAUSE
  END IF

  MESSAGE "Check that the Reprogramming light is blinking."
  WAIT 1.0
  MESSAGE "Resume to continue." PAUSE

  EVERY 5.0 WITHIN 30.0
    MESSAGE "AGHF Experiment Running..."
  END EVERY

CLOSE SUBSEQUENCE Automatic_Run


SUBSEQUENCE Start_Malfunction_Procedures

  START Check_Failure_And_EP_Off_lt
  START Check_Setup_Sample_lt
  START Check_Chamber_Open_lt
  START Check_Load_PROM_lt
  START Check_Power_And_EP_On_lt

CLOSE SUBSEQUENCE Start_Malfunction_Procedures


SUBSEQUENCE Stop_Malfunction_Procedures

  STOP Check_Failure_And_EP_Off_lt
  STOP Check_Setup_Sample_lt
  STOP Check_Chamber_Open_lt
  STOP Check_Load_PROM_lt
  STOP Check_Power_And_EP_On_lt

CLOSE SUBSEQUENCE Stop_Malfunction_Procedures

SEQUENCE Check_Failure_And_EP_Off_lt
```

```
  WHENEVER ((Failure_Light = ON) AND (EP_Off_Light = On)) THEN
   MESSAGE "Warning: The Failure Light is on and the EP Off Light is On."
   MESSAGE "Report light status to POCC."
   WAIT 2.0
   MESSAGE "Flip Power On/Off switch off."
   WAIT 1.0
   MESSAGE "Resume to continue." PAUSE

   WAIT 10.0

   MESSAGE "Flip Power On/Off switch on."
   WAIT 1.0
   MESSAGE "Resume to continue." PAUSE

   MESSAGE "Check with POCC to report light status."
  END WHENEVER

CLOSE SEQUENCE Check_Failure_And_EP_Off_lt


SEQUENCE Check_Setup_Sample_lt

  WHENEVER (Setup_Sample_Light = OFF)
   MESSAGE "Warning: The Setup Sample Light is off."
   MESSAGE "Notify POCC."
  END WHENEVER

CLOSE SEQUENCE Check_Setup_Sample_lt


SEQUENCE Check_Chamber_Open_lt

  WHENEVER (Chamber_Open_Light = ON)
   MESSAGE "Warning: The Chamber Open Light is on."
   MESSAGE "Notify POCC."

   --really more involved than this
   MESSAGE "Stopping Experiment."
   HALT AGHF_Experiment
```

```
 END WHENEVER

CLOSE SEQUENCE Check_Chamber_Open_lt


SEQUENCE Check_Load_PROM_lt
 WHENEVER (Load_PROM_Light = OFF)
  MESSAGE "Warning: The Load Prom Light is off."
  MESSAGE "Notify POCC."
 END WHENEVER

CLOSE SEQUENCE Check_Load_PROM_lt

SEQUENCE Check_Power_And_EP_On_lt
 WHENEVER (Power_Light = OFF)
  MESSAGE "Warning: The Power Light is off."
  IF (EP_On_Light = ON) THEN
   MESSAGE "EP On light is on.  Notify POCC of faulty Power light."
  ELSE
   MESSAGE "The EP On light is off. The power failed.  Notify POCC."
  END IF
 END WHENEVER
CLOSE SEQUENCE Check_Power_And_EP_On_lt

SUBSEQUENCE Initialize_Variables
 SET Exp_Vent_Bleed_Vlv TO CLOSED
 SET Exp_Vent_Vlv TO OPEN
 SET Process_Finished_Light TO OFF
 SET Hot_Sample_Light TO OFF
 SET Setup_Sample_Light TO ON
 SET Reprogramming_Light TO ON
 SET Preheating_Phase_Light TO ON
 SET Vacuum_Valves_Open_Light TO ON
 SET Operations_Light TO ON
 SET Failure_Light TO OFF
 SET EP_Off_Light TO OFF
 SET EP_On_Light TO OFF
 SET Chamber_Open_Light TO OFF
 SET Load_PROM_Light TO ON
 SET Power_Light TO ON
```

```
  CLOSE SUBSEQUENCE Initialize_Variables


CLOSE BUNDLE AGHF_Experiment
```

References

[1] Nyberg, K. A., *The Annotated Ada Reference Manual*, Grebyn Corporation, Vienna, VA, 1991.

[2] C. S. Draper Laboratory, *SSP 3059 Timeliner User Interface Language*, Revision C, C. S. Draper Laboratory, Cambridge, MA, 1995.

[3] Command Corporation, Inc., *IN^3 Voice Command User's Guide for SPARCstation*™, Version 2, Command Corporation, Inc., Atlanta, GA, 1994.

[4] Entropic Research Laboratory, Inc., *Personal TrueTalk™ User's Guide*, Entropic Research Laboratory, Inc., Washington, DC, 1995.

[5] Kelley, A. and Pohl, I., A Book on C, Second Edition, The Benjamin/Cummings Publishing Company, Inc., Reading, MA, 1990.

[6] Ousterhout, J. K., *Tcl and the TK Toolkit*, Addison-Wesley Professional Computing Series, Addison-Wesley Publishing Company, Reading, MA, 1994.