`sniffle`: A Simple Networked Interface to the Full Functionality of Laboratory Equipment

Terence Kelly
E-mail: `tpkelly@cs.princeton.edu`
WWW: `http://www.cs.princeton.edu/~tpkelly/`

September 14, 1995

Abstract

This paper describes the `sniffle` project and system. `sniffle` enables World Wide Web clients to remotely control the most important functions of an analog/digital I/O board installed in an IBM-compatible PC and interface with a digital oscilloscope attached to this server PC. The Web client can

- specify digital and analog signals to be output by the I/O board,
- obtain digital and analog inputs read by the board,
- obtain waveforms from the ditital oscilloscope and, more generally,
- control all of the capabilities of the digital oscilloscope.

At present, the `sniffle` server is configured to control a platform which pans and tilts a digital camera in response to Web client commands issued via an imagemap. The camera provides real-time video feedback to remote clients via Cornell University's CU-SeeMe teleconferencing system.

`sniffle` differs from other work in the field of Internet remote control in that it is a modular, general-purpose *tool* rather than a tightly integrated special-purpose system. It was completed in a short time at low cost in under two thousand lines of code. `sniffle` thus demonstrates that remote control via the World Wide Web is not an inherently costly or complex problem.

This project report is available in HTML form at URL `http://www.cs.princeton.edu/~tpkelly/sniffle/`.

1 Preface

The author developed the `sniffle` system in the summer of 1995 under the supervision of Professor Michael Littman of Princeton University's Department of Mechanical and Aerospace Engineering. Although parts of this report necessarily deal with the details of Dr. Littman's lab and the Princeton University computing climate, most of what follows will be of very general interest.

This paper is intended to serve many purposes for several distinct audiences. It provides a detailed introduction to the `sniffle` project and describes its place in the field of Internet remote control. It also contains a guide for end-users of `sniffle` a detailed explanation of how it works for anyone interested in modifying or extending it, and ideas for improving the system. Complete listings of all of the Visual Basic code at the core of `sniffle` as well as the author's project log are provided in appendices.

2 Introduction

This section describes the motivation, history and system architecture of the `sniffle` system and discusses how it relates to other Internet remote control projects.

Remote Control on the Internet

Ours is not the first project to use the Internet for remote control. It is beyond the scope of this document to describe other projects in detail. Fortunately, descriptions are readily available on the Web. Useful Web pages containing pointers to many such projects are available at the following locations:

- Yahoo
- `http://www.indstate.edu/msattler/sci-tech/comp/misc/remote-control.html`
- `http://www.eia.brad.ac.uk/mark/fave-inter.html`

These projects vary greatly in complexity. Even the simplest of them often involve "live" video feedback. They range in sophistication from a simple scheme to monitor a distant coffee pot to an elaborate project which allows people from all over the world to share control of an industrial robot arm. All of these other projects, however, differ from `sniffle` in one crucial respect: they are without exception primarily *demonstrations* rather than *tools.*

By contrast, the simple `sniffle` system is flexible enough to be adapted to a wide variety of applications. Anyone with an IBM PC and an RTI-815 I/O board can easily customize `sniffle` to their system and thus obtain control of the most important features of the board via the Web. All that need be done in order to obtain this powerful capability is install WinHTTPD (very easy) and tweak a bit of Visual Basic code (easy).

A Brief History of `sniffle`

The current version of `sniffle` differs fundamentally from the project that Dr. Littman and I initially envisioned. One can trace the evolution of our thinking in my project log, which is included in an appendix. A more succinct summary of our early thinking can be found in our earliest project description, available at `http://www.cs.princeton.edu/ tpkelly/webrc.html`.

We began with an needlessly narrow vision of a simple, practical solution to a particular problem. Many of Dr. Littman's lab demonstrations involve delicate arrangements of cumbersome devices. It is inconvenient to disassemble and reassemble the demonstrations for one-time showings, e.g. to classes of students. It is also impractical to crowd a large class into a small laboratory in order to present a demonstration. Dr. Littman would like to set up his experimental apparatus in his lab and show it to audiences anywhere without much fuss. To achieve this goal we decided to use two independent systems, one to control an I/O board which in turn would control the actual lab demo, and another to handle video feedback.

It gradually dawned on us that what we really wanted was a general-purpose tool that would allow us to interact with an I/O board in a PC at the lab from anywhere in the world. We realized that we could wrap more general capabilities into the system, e.g. the ability to communicate with a digital oscilloscope via a serial connection. This was the insight that turned a one-time throwaway "gee-whiz" demonstration project into an extensible, reusable remote control system.

System Architecture

A picture's worth a thousand words, so it's best to begin with a diagram of the current state of the `sniffle` system. The next version of this report will include one here.

Ingredients

The "ingredients" used in the first version of `sniffle` are listed below, along with vendor information.

Web Server

- IBM PC AT upgraded to 486 processor, 8 MB RAM.
- Bob Denny's WinHTTPD server. Runs *only* with Windows 3.1, *not* Windows for Workgroups, 95, NT, etc.. `http://www.city.net/win-httpd/`.
- Analog Devices RTI-815. (617) 461-4111.
- Two Airpax K82231 stepper motors (obsolete). Similar motors are available from the distributor, Sager Electronics, (800) 724-37800. (Yes, that phone number is correct.)
- Two SAA 1027 controller chips for stepper motors. Also available from Sager Electronics.
- Tektronix 222 digital oscilloscope. (800) 835-9433.
- Microsoft Visual Basic Professional Edition. The Professional Edition is necessary because it contains a control for serial communications which is not included in the Standard Edition. Furthermore Bob Denny, the author of the WinHTTPD server, recommends Visual Basic for cgi-win back-ends.
- `WEBRTI.EXE`, the Visual Basic executable at the heart of the `sniffle` system.

CU-SeeMe Server

- Macintosh PowerBook Duo 280c.
- Connectix QuickCam digital camera.

The CU-SeeMe system uses the same software for both client and server; see the list of CU-SeeMe software packages below.

Web Client

Netscape is recommended. The latest versions for a variety of platforms are available at `http://home.netscape.com/comprod/mirror/index.html`.

CU-SeeMe Client

CU-SeeMe clients are available for MS-Windows, Macintosh, and Power Macintosh at `http://cu-seeme.cornell.edu/get_cuseeme.html`

An X-windows package called `nv` can allegedly receive CU-SeeMe video, but we haven't been able to get this working yet. The `nv` distribution is available at `ftp://parcftp.xerox.com/pub/net-research/`.

3 User's Guide

This section explains how to use `sniffle` to remotely control the most important functions of the analog/digital I/O board installed in the server PC and communicate with the digital oscilloscope. Refer to the appropriate Analog Devices and Tektronix manuals for further information on the RTI-815 I/O board and the 222 digital oscilloscope.

What `sniffle` Can and Can't Do

The most important component of the `sniffle` system is a *command interpreter* which receives commands from the Web client via the WinHTTPD server's cgi-win interface. If you're unfamiliar with the jargon, think of it this way: we use the Web to move a string of commands from the client to an interpreter program at the server. The interpreter "executes" these commands by calling routines that communicate with the digital oscilloscope and I/O board connected to the server. These components in turn gather data from the environment by measuring the voltages on input lines and affect their environment by setting voltages on output wires. As it is currently configured our RTI-815 board provides eight bits of digital output, eight bits of digital input, sixteen analog input channels, and two analog output channels.

As the command interpreter goes about its business, it records information that will eventually be returned to the client. This information includes voltage values detected by the I/O board in response to "input" commands, waveforms taken from the oscilloscope, and error messages. After the last command is interpreted, these data are made available to the client.

The "language" or set of macro commands recognized by the interpreter gives the client control of the most important analog and digital input and output functions of the I/O board. It also allows the client total control over the digital oscilloscope. There's nothing you can do to the oscilloscope when you're standing in front of it that you can't do from a Web client using `sniffle`. Finally, `sniffle` gives the client the ability to pause the interpreter for a while, iterate arbitrary commands a specified number of times, and run batches of commands stored at the server end.

One useful and important thing that the `sniffle` system *cannot* do is respond to run-time events. The command interpreter blindly executes one command after another, oblivious even to the readings it obtains from its environment via the oscilloscope and the I/O board. The `sniffle` command language is not a true programming language; it lacks the ability to branch or to evaluate conditions.

Getting Started

"Hello World"

The easiest way to become familiar with `sniffle` is to use it, so we'll examine what happens when we write and submit the world's most famous program to the interpreter from a Web client. The first thing to do is load the `sniffle` program form into your Web browser. The URL is `http://piper.princeton.edu/webrti.htm`. This form contains a "textarea" containing a sample program. If we delete the sample program completely, enter the string "`hello world`" in the textarea, and then click on the "Submit" button, we send that string to the command interpreter.

If all goes well at the server end, it won't be long before we receive a reply. We should get a document containing a link to the session log that records the results of the command we submitted. We're interested in the session log, so we follow this link. It should look something like this:

```
BEGIN PROGRAM 02:04:44 9/12/95
RTI Initialization:
COM Initialization: READY;

0/1 : HELLO WORLD : Unrecognized command passed to routine Exec

END PROGRAM 2:4:45 9/12/95
```

The first and last lines of this output record the times when the interpreter program (a Visual Basic program called `WEBRTI.EXE`) started and terminated. The second and third lines record the results of I/O board and serial port initialization, respectively. In the next line the interpreter "echoes" our command, indicating that the command was received. In general, for every command interpreted by `WEBRTI.EXE` one line is added to the session log. This line consists of three parts delimited by colons.

The first part is a number which records the "depth" of the command followed by a slash followed by the command number. In our example, the depth of our command is zero, and we submitted one command, so in our session log we have "`0/1`." The second part of the line is the command itself, and the third part contains either values returned (in the case of input commands, described below) or error messages. The `sniffle` command interpreter is robust and functional, but it isn't very sociable. It didn't recognize our greeting and says so.

Server Locking

We don't ever want two clients operating the I/O board or digital oscilloscope simultaneously, so the server has a primitive notion of "ownership" implemented by a crude but effective file-based locking system. Whenever `WEBRTI.EXE` is executed by the WinHTTPD server, one of the first things it does is check to see if a lock file exists. If so, a message to the effect that the system is locked is returned to the client. The client then has the opportunity to "break" the lock by forcibly deleting the lock file. This should only be done if it is certain that the server is erroneously locked (e.g. as a result of a server or `WEBRTI` crash).

If `WEBRTI.EXE` does not find a lock file at startup time, it creates one and then deletes just before it terminates. In principle, therefore, it shouldn't be possible for two Web clients to confuse the server by issuing conflicting commands.

Data Feedback

Data feedback to the client takes two forms. The first and simplest consists of analog and digital input values detected by the I/O board. These are returned to the client in the session log. So if we tell the interpreter to read the whole byte of digital input by sending it the "`din`" command, we get back something like this:

```
BEGIN PROGRAM 02:37:03 9/12/95
RTI Initialization:
COM Initialization: READY;

0/1 : DIN : value = 0 at 2:37:04 9/12/95

END PROGRAM 2:37:04 9/12/95
```

This indicates that all eight bits of digital input were "low" at the time the command was executed.

The second form of data feedback consists of oscilloscope dumps. These are generated by the "`tek`" command (see below for details). After a program has run its course, if instead of following he link to the session log we follow the link to "Tektronix Dumps" we will obtain a list of links to individual waveforms generated. Each waveform is a text file with a `.WFM` extension. The WinHTTPD server is configured to label these files with MIME type "text/waveform" when returning them to the client. MS-Windows clients can be configured to spawn the Tektronix DOS utility `VIEW.EXE` upon receipt of files with this MIME type. Alternatively, the client can load the `.WFM` file to local disk (in Netscape this is done by holding down the shift button while clicking on the link). The waveform file can then be decoded, displayed or manipulated at the client. The format of `.WFM` files is explained in Tektronix documentation.

Interpreter Commands

The macro language understood by the `sniffle` interpreter is very simple, so there aren't many rules to remember. The interpreter is not case sensitive, so it doesn't distinguish between "`hello world`" and "`HeLLo wORLd`." A comment is anything from a pound sign (`#`) to the next end-of-line or semicolon. The only absolute rule is: never place a semicolon inside a comment, or an error will result, because the interpreter thinks the comment ends at the semicolon.

Output

`sniffle` provides three commands for controlling the digital output functions of the RTI-815 I/O board on our server PC, listed below. See the Analog Devices *RTI-800/815 User's Manual* for information on pin assignments.

- `XAOT <channel> <value>` sets the voltage on the specified analog output channel to the given value. The `channel` parameter must be either 0 or 1, and `value` must be a number in the range -9.99 to 9.99.
- `DOT <value>` produces digital output. The `value` parameter is a decimal number in the range 0-255 or a hex number in the range H0-HFF. You must precede hex numbers with an "H."
- `DOTB <bit> <1|0>` writes zero or one to the specified `bit` of the digital output, where `bit` is a number in the range 0-7.

The `DOTB` command is useful because it is sometimes handy to be able to toggle an individual digital output bit without worrying about or affecting the other digital output bits. The pan-and-tilt control scheme, for instance, relies on `DOTB` to change the digital output bits we use for controlling our camera platform without knowing or changing the other digital output bits.

Input

Two commands allow the client-end `sniffle` user to measure the voltages on wires connected to the I/O board of the server PC:

- `XAIN <channel>` records the voltage on the specified `channel` and returns this value to the user in the session log. The `channel` parameter is a number in the range 0-15.
- `DIN` reads the byte of digital input and records a hex-encoded value in the range 0-FF in the session log.

Digital Oscilloscope

Our server PC is currently connected to a Tektronix 222 digital oscilloscope via a serial connection. Through the commands below the `sniffle` user can control literally every feature and function of the oscilloscope. For more information on the strings recognized by the oscilloscope, see the Tektronix manual entitled *222 RS-232 Interface Guide*.

- `TEK SEND <delay> { <string> }` sends the string enclosed between curly braces to the Tektronix scope and waits `delay` seconds before attempting to read a reply from the scope. *Any* command whatsoever can be sent to the scope using this command; the other two `TEK` commands are special cases which include a few user friendliness features. It is the user's responsibility to choose the `delay` interval wisely (ten seconds will almost certainly be long enough for any single command). The reply returned by the oscilloscope is returned to the user in the session log, along with an English decoding of any error codes returned by the scope.

- `TEK GET` grabs both waveforms from the oscilloscope and stores them in files on the server. The client can obtain these waveforms after the `WEBRTI.EXE` program has terminated.
- `TEK FP? <logfp>` obtains from the oscilloscope a string encoding the current state of `logfp`, where `logfp` is usually "`acq`," denoting the acquisition system. A detailed listing of the current `logfp` settings is returned to the user in the session log. This is how to find out all there is to know about the scope's settings.

Miscellaneous

- `DELAY <n>` causes the `WEBRTI.EXE` program to pause for `n` seconds.
- `LOOP <n> { <commands> }` causes the given `commands` to be executed `n` times. The commands must be enclosed in curly braces. Nested loops don't work, but otherwise an arbitrary list of semicolon-delimited commands can be used.
- `SOURCE <filename>` executes commands contained in a file on the server. These files are stored in directory `C:\HTTPD\HTDOCS\WEBRTI\` and have `.RTI` extensions. This extension is *not* part of the `SOURCE` command. So "`SOURCE FOO`" is correct but "`SOURCE FOO.RTI`" is wrong.

The `SOURCE` command is worth discussing in further detail. It provides a powerful mechanism for the client-end user to invoke batches of commands stored at the server end. Furthermore, `SOURCE` commands *can* be nested. Let's consider an example.

We will use two `.RTI` files in `C:\HTTPD\HTDOCS\WEBRTI\`: `TEST02.RTI` and `TEST03.RTI`. `TEST02.RTI` looks like this

```
#
#  Simple file to test ability of WEBRTI to handle "source" commands.
#
DOT 2; DELAY 2
DOT 3; DELAY 2

SOURCE TEST03

DOT 6; DELAY 2
DOT 7; DELAY 2
```

and `TEST03.RTI` looks like this

```
DOT 4; DELAY 2
SOURCE NOT-HERE
DOT 5; DELAY 2
```

We go to `http://piper.princeton.edu/webrti.htm` and enter the following:

```
dot 1; delay 2
source test02
dot 0
```

If we submit this set of commands from the vicinity of the server PC, we might be able to watch the digital output slowly count from zero to seven as the commands are executed. After our "program" has run, we get the following feedback in the session log:

```
BEGIN PROGRAM 02:43:06 9/13/95
RTI Initialization:
COM Initialization: READY;

0/1 : DOT 0 :
0/2 : DELAY 2 :
0/3 : DOT 1 :
0/4 : DELAY 2 :
0/5 : SOURCE TEST02 :
   1/6 : DOT 2 :
   1/7 : DELAY 2 :
   1/8 : DOT 3 :
   1/9 : DELAY 2 :
   1/10 : SOURCE TEST03 :
      2/11 : DOT 4 :
      2/12 : DELAY 2 :
      2/13 : SOURCE NOT-HERE : file empty or nonexistent:
                      C:\HTTPD\HTDOCS\WEBRTI\NOT-HERE.RTI
      2/14 : DOT 5 :
      2/15 : DELAY 2 :
   1/16 : DOT 6 :
   1/17 : DELAY 2 :
   1/18 : DOT 7 :
   1/19 : DELAY 2 :
0/20 : DOT 0 :

END PROGRAM 2:43:23 9/13/95
```

The `sniffle` interpreter keeps track of the "depth" of each command it executes. The commands we submitted to the server are at "depth" zero, so in the session log they are labelled 0/1, 0/2, etc.. The commands in `TEST02` are at "depth" one, so the number before the slash is 1. `TEST02` sources `TEST03`, so the commands in `TEST03` are at depth 2. Lines in the session log are indented according to the depth of the commands to which they correspond.

Administering the Server

Troubleshooting

If something unexpected happens, check the program you submitted (by clicking on your browser's "back" button) to make sure it is correct. Did you issue a command like "`LOOP 60 { DELAY 60 }`", which will take an hour to execute? Did you `SOURCE` a file which `SOURCE`s another file which `SOURCE`s another file etc.?

The interpreter does a pretty good job of returning error messages in response to bogus commands, and the server tries to trap run-time errors intelligently and report on them. So the second place to look for problems is in the session log. If the server does not respond to a program, and if you're sure that the program is correct, it is possible that the network is down or that the server has crashed. In the latter case the server will have to be manually re-started from the console. One last place to look for clues is the `WEBRTI` log, which lives in the same directory as the WinHTTPD access log. It records every start and termination of `WEBRTI` and the commands issued to it.

Most Web browsers create local caches for efficiency, and browsers often retrieve documents from the cache rather than the remote source in the mistaken belief that the cache contains an up-to-date document. This can cause trouble for the `sniffle` user in certain situations, so consider clearing your browser's cache and disabling caching.

I doubt that any bugs remain in the `WEBRTI` program, but if you find one, mail `tpkelly@cs.princeton.edu` to report them. Be sure to include a detailed description of the problem. The easiest way to send me mail is to use my mail submission form: `http://www.cs.princeton.edu/~tpkelly/`.

Good Housekeeping

Each set of commands submitted to the interpreter generates a separate session log, and these log files can be quite large. I once generated a 60-kilobyte session log with a one-line command. We don't want these session logs piling up on the server, so if there are too many of them the client is given an opportunity to delete all of them. Be sure to save any important session logs before doing this!

The WinHTTPD server communicates with back-end programs (like `WEBRTI.EXE`) via temporary files in server directory `C:\WINDOWS\TEMP\`. These temp files are normally deleted by the server after they are no longer needed, but in practice things can go wrong (e.g. the server can crash), and so these temporary files can accumulate. Purge them periodically.

Finally, it is a good idea to periodically de-fragment the server's hard drive to improve performance.

Pan-Tilt Controller

At the time of this writing, the `sniffle` system is configured in such a way that the low four bits of the server's digital output control a pair of stepper motors which pan and tilt a digital camera. Although strictly speaking the camera platform controller is not an integral part of `sniffle`, it is expected to be used extensively with many applications of the `sniffle` system and therefore deserves mention here.

The pan-tilt control panel is at URL `http://piper.princeton.edu/webrti/pantilt.htm`. It provides a simple and self-explanatory interface to the pan-and-tilt platform which controls the orientation of the digital camera used to provide video feedback to the client. Clicking on the small arrows of the control panel will rotate the camera roughly five degrees; the big outer arrows cause fifteen degrees of rotation.

4 Under the Hood

This section describes in detail how the interpreter at the core of `sniffle` works. To learn about other aspects of the `WEBRTI.EXE` program, read subroutine `Main` of module `webrti.bas`, provided in an appendix. It's under two pages of thoroughly commented code, and it should be clear to anyone who understands Visual Basic. It is beyond the scope of this document to explain the off-the-shelf components of `sniffle`, e.g. the WinHTTPD server. Refer to the appropriate documentation to learn more about them.

This section assumes a basic understanding of HTML, HTTPD servers, the WinHTTPD cgi-win interface, the RTI-815 I/O board, and the Tektronix 222 digital oscilloscope.

Caveat: in the Princeton Computer Science department, students in programming courses are taught never to trust a program's documentation. In real life, the code behind a project and the code's documentation quickly fall out of sync. It usually isn't long before the documentation becomes, in the words of Bjarne Storstrup, "a fairy tale." Every reasonable effort has been made to ensure the accuracy of the report that you are now reading, but if you really want to understand how `sniffle` works, read the code. You have been warned.

Getting Started

In order to better understand the file-based communication scheme used by the WinHTTPD cgi-win interface, I wrote a simple cgi-win program which can help clarify this crucial interface. The URL is http://piper.princeton.edu/cgi-win/web_back.exe. When a Web client accesses this URL with or without submitting form data to it, the WEB_BACK.EXE program returns a listing of all of the interesting stuff that *it* got from the WinHTTPD server. Read Bob Denny's excellent documentation on the WinHTTPD server and play with my little WEB_BACK.EXE program in order to understand how the cgi-win interface works.

The sniffle Interpreter

This section describes the operation of the interpreter at the core of the sniffle system. It assumes a basic knowledge of the WinHTTPD server, the cgi-win interface, and Visual Basic.

The easiest way to learn how the interpreter works is to follow a string of commands on its journey from the client's Web browser through execution at the server and back to the client, in the form of diagnostic output. The discussion that follows will be much clearer if you simultaneously read the Visual Basic code listed in an appendix. Start reading in module webrti.bas at subroutine Main.

The process begins when a client-end user enters a set of sniffle commands in his/her Web browser as described in the "User's Guide" section of this document. The Web client URL-encodes the contents of the and POSTs it to the WinHTTPD server. Alternately, the client may use the somewhat outmoded GET method, appending a query string to the URL it submits to the server; this method is used with clickable imagemaps. In either case a URL-encoded string is sent to the WEBRTI.EXE cgi-win program at the server end.

The URL-encoded string consists of a series of key-value pairs, and one of the keys must be "commandstring." WEBRTI.EXE looks for this key in two places. It first checks GET-method query strings, then POST-method data bodies. It extracts the value corresponding to key "commandstring," decodes it, strips out comments and translates newlines to semicolons, then passes the resulting string along to subroutine Interp.

Interp splits the command string on semicolons and passes each individual command to subroutine Exec. Exec decides what sort of command it has received (e.g. TEK, DIN, etc.) and passes the buck to subroutines which execute the appropriate Visual Basic commands. These subroutines return strings containing either error messages or, in the case of input commands, feedback from the oscilloscope or I/O board, which Exec appends to a global log string. This string is eventually returned to the client as part of the session log.

Camera Pan-Tilt Controller

When a client-end user clicks on a "button" of the pan-tilt control panel imagemap, the Web client submits the coordinates of the image pixel which was clicked to the IMAGEMAP.EXE program on the server using the GET method. The first coordinate ("x") represents the distance from the left border of the image that was clicked and the second ("y") refers to distance from the top edge. The IMAGEMAP.EXE program checks the appropriate map file to decide what to do. Here's the map file:

```
default /webrti/default.htm

rect /cgi-win/webrti.exe?commandstring=source+u&rf=pantilt.htm  136,88  168,120
rect /cgi-win/webrti.exe?commandstring=source+d&rf=pantilt.htm  136,168 168,200
rect /cgi-win/webrti.exe?commandstring=source+l&rf=pantilt.htm  96,128  128,160
rect /cgi-win/webrti.exe?commandstring=source+r&rf=pantilt.htm  176,128 208,160

rect /cgi-win/webrti.exe?commandstring=source+bu&rf=pantilt.htm 136,48  168,80
rect /cgi-win/webrti.exe?commandstring=source+bd&rf=pantilt.htm 136,208 168,240
rect /cgi-win/webrti.exe?commandstring=source+bl&rf=pantilt.htm 56,128  88,160
rect /cgi-win/webrti.exe?commandstring=source+br&rf=pantilt.htm 216,128 248,160
```

If the coordinates submitted by the client do not fall within any of the rectangles defined in this file, a default message (essentially "you missed") is retured from the file specified on the first line of the map file above. But if they fall within the rectangle with upper-right coordinates 176,128 and lower-right coordinates 208,160 the `IMAGEMAP.EXE` program activates `WEBRTI.EXE` and passes to it a query string. The query string, `commandstring=source+r&rf=pantilt.htm`, contains two key-value pairs. The first tells `WEBRTI.EXE` to execute the batch of commands contained in file `R.RTI`. The second tells `WEBRTI.EXE` to return file `PANTILT.HTM` to the client instead of the usual diagnostic output. This means that the client gets back the same camera-control form that he/she started with, which is better than having the control panel disappear every time it's clicked.

Some details are omitted from the discussion above. For the full story on using clickable imagemaps with the WinHTTPD server, read the documentation that accompanies the server.

5 Going Further

This section describes ways in which `sniffle` can be extended and improved.

The first "production version" of the `sniffle` system is a reasonably complete general-purpose tool for remotely controling electronic equipment. It should therefore be useful for a wide range of applications in Dr. Littman's laboratory and perhaps beyond. It is best to apply the system to a number of different tasks and use it to the limits of its ability before adding features or extensions. Only after exhausing its existing capabilities should one attempt to tinker with it; if it ain't broke, don't fix it.

There may come a time, however, when the initial version of `sniffle` falls short of an application's specifications. The system is designed in a fairly flexible, extensible, modular way. It should be relatively easy to extend `sniffle` or to steal parts of it for use with other systems.

Prerequisites

In order to modify or extend the `sniffle` system without vandalizing it, one must have a firm background in software systems and a good working understanding of a number of important issues. Minimal formal "prerequisites" include COS 217 and COS 228; COS 333 and COS 461 would be nice, as would experience with complex software projects involving thousands of lines of code. Before embarking on any modifications to `sniffle` become familiar with all relevant components of the system (e.g. HTML, HTTP, cgi interfaces, the WinHTTPD server, serial communications, the Tektronix digital oscilloscope, I/O boards, etc.). A good way to learn about these issues is to complete a "warm-up" exercise; see the next section for ideas.

Most importantly, an orderly and methodical approach to development is essential. It is possible to hack together a small system (on the order of hundreds of lines of code) without writing clean, modular, well-documented, versioned code, but the "quick and dirty" approach does not scale well as project size and complexity increase and as re-usability and maintainability become important project goals. The point of the `sniffle` project is *not* to produce a one-time throw-away Rube Goldberg apparatus which merely accomplishes some special task. Rather, the point is to develop a robust, extensible, general-purpose *tool* whose usefulness transcends any particular task to which it is applied.

A project of this size requires disciplined software habits, foremost of which is version control. I have installed a DOS version of GNU RCS on our current server PC and have used it with the `sniffle` source code. At least two commercial version control utilites for Visual Basic are available, as are several documentation aids. These might well be worth the price. Select and learn a version control scheme before modifying any source code. Making regular (i.e. daily) backups is another essential habit. CIT's ADSM backup system might provide a long-term solution to the latter issue. Contact a company called "VBxtras" at (800) 788-4794 or (404) 952-6356 for more information on version control utilities for Visual Basic.

Local Resources for Developers

During the course of the `sniffle` project I installed a number of useful utilities on the server PC in Dr. Littman's lab. Many of these should be useful to anyone trying to modify or extend `sniffle`.

Directory `C:\GNU\BIN\` contains DOS versions of a number of useful Unix utilities, including `grep`, `egrep`, `less`, `diff`, `perl`, the `rcs` family of version control programs, `sort`, `unzip` and `zoo`. Directory `C:\BIN\` contains `PKUNZIP.EXE`. The DOS utilities provided by Tektronix for grabbing and converting oscilloscope traces are in `C:\222UTIL\`.

By far the most important of these are `rcs` and `perl`. Back-end scripts can be written for the WinHTTPD server in `perl`. The author of WinHTTPD the use of DOS back-ends, because they must be run in error-prone virtual DOS machines under Windows. However, `perl` is such a powerful language that in certain applications its advantages will outweigh this consideration. The `rcs` programs make it possible to maintain versioned copies of software modules on a PC. In many ways it's better to use the Princeton Unix system for version control, because this automatically makes a very safe set of off-site backups, but uploading files to arizona can be a hassle so it's nice to have a choice.

I also installed a number of useful Windows utilities. Gnuplot is in the "Applications" folder; the executable is in `C:\GNU\GNUPLOT\`. GIF Construction Set, which is useful for creating complex GIF files, (`C:\GIFCON\GIFCON.EXE`) has its own folder. Several useful graphics applications live in `C:\GRAPHICS\`; the icons are in folder "Terry's Apps." Yet another graphics application, Graphic Workshop for Windows, is in `C:\GWSWIN\`.

Warm-up Exercises

It would be nice to have a simple, user-friendly graphical Web interface to the Tektronix 222 digital oscilloscope. Producing such an interface would force the developer to learn the basics of the WinHTTPD server and its cgi-win interface, the RS-232 serial interface to the oscilloscope, Visual Basic, and a few modules of my `sniffle` source code. Developing such an interface would be a good medium-sized warm-up exercise for anyone considering extending `sniffle`.

Tying Up Loose Ends

Much of the `sniffle` source code, listed in an appendix below, leaves room for improvement. For instance, aspects of system configuration (e.g. the slot in which the I/O board is installed and the COM port used for serial communications) are hard-coded. Things like this should really be set in a global system configuration file.

Another loose end involves the firmware in our Tektronix 222 scopes. It needs to be upgraded. See my project log, included in an appendix, for details.

Extending the Interpreter

Adding a command to the macro language recognized by the interpreter at the core of the `sniffle` system can be a very simple matter. A single function in module `WEBRTI.BAS`, namely `Exec`, evaluates individual commands. By adding another `Case` to the `Select Case` statement at the heart of this function one can extend the `sniffle` macro language. Read about the inner workings of `sniffle` in the "Under the Hood" section and stare at the `WEBRTI.BAS` source. It should be obvious how to proceed.

Graphical Feedback

At the time of this writing both of the methods by which `sniffle` provides graphical data feedback to the client leave much to be desired. One method is to configure MS-Windows clients to spawn the Tektronix 222 DOS utility `VIEW.EXE` upon receipt of files with MIME type "text/waveform" and to configure the server to label outgoing `.WFM` files with this MIME type. Our second method is for the client to download waveforms to disk and then use a custom-written Matlab script to decode the `.WFM` file into a form intelligible to Matlab, which can then produce graphics.

Both of these methods are problematic because they are clumsy and insufficiently general. The first method only works with MS-Windows clients and requires client-end users to reconfigure their Web browsers. Much of the world uses X-Windows or Macintoshes, and reconfiguring a browser is a hassle beyond the abilities of most Web surfers. The second method requires even more special expertise than the first, as well as a custom-written Matlab script.

For these reasons it would be nice to have the `sniffle` server generate `.GIF` files from waveform data (or, for that matter, from the I/O board's analog and digital readings). This could be done in a two-stage process: waveform files are converted to `ppm` ("portable pixmap") format by the `sniffle` program, and the resulting `ppm` data is converted to `.GIF` format by `ppmtogif`. The `ppmtogif` program is available on the Princeton Unix system and can probably be ported to DOS with minimal fuss. Read "`man ppm`" on the Unix system for more information on the portable pixmap file format. It's a very simple least-common-denominator format that can be understood in under five minutes. It is possible that many Web clients are configured to deal with `.PPM` files directly; I have not investigated the issue. The Windows graphics viewer `LVIEWP` is installed on our server PC and can be used to display `.PPM` files and can presumably be used as a client-end helper application.

Another option is to generate PostScript at the server. PostScript viewing utilities are available for most Web clients. PostScript can be generated directly, or perhaps with the help of a utility such as Gnuplot provided that the utility can be made to run in "batch mode." The MS-Windows version of Gnuplot is installed on our server PC.

Yet another option similar to the current Matlab scheme, but which does not require custom software at the client end, would be to generate a generic plot file at the server and use something like Gnuplot at the *client* end to view the file. Gnuplot is available for X-windows and MS-Windows.

Replacing the Interpreter

Many interpreted languages feature an "eval" or "exec" command which allows a program to execute an arbitrary string (including one that is obtained or built by the program *at run-time*) as though the string were part of the program itself. Perl, Tcl/Tk and the Unix shells all have this feature. Visual Basic, however, lacks it. If the core of the `sniffle` system were written in a language which supported eval, the client could submit ordinary programs in this language and the server could simply eval them. This would provide a much more elegant solution than the current scheme and would give the client-end user a much more powerful way to control the server-end I/O board. The client would be able to submit a true *program* to the server, rather than merely a series of commands. Programs could contain conditional statements and variables and could therefore cause the I/O board's output to *respond* to the board's inputs.

To the best of my knowledge, however, the subroutine libraries supplied by Analog Devices and other I/O board manufacturers do not support any language with eval/exec. It might be feasible to hack a way around this problem by using e.g. a Perl script as the cgi back-end to a Web server and having the script invoke small supplementary programs written in C or Pascal to accomplish low-level board control tasks. A DOS version of Perl is available and can be used with WinHTTPD. Beware, however, that the author of WinHTTPD strongly discourages the use of DOS programs as cgi back-ends, as these must be run in buggy virtual DOS machines under Windows.

The Right Thing To Do

The right solution probably involves extending an interpreted language (e.g. Perl or Tcl/Tk) by adding commands that control the I/O board. Rumor has it that it's fairly easy to roll your own C code into Tcl/Tk in a Unix environment, and this sort of splicing might be equally straightforward in on a PC. Tcl/Tk is available for MS-Windows and a wide variety of other platforms. Read all about it at `http://cuiwww.unige.ch/eao/www/TclTk.html`. The more I think about it, the more I wish I'd thought of this two months ago. (Caveat: serial communications are probably not nearly as easy in Tcl/Tk as in Visual Basic.)

Changing the Communications Component

For the next few years the Internet and Web will be painfully slow. If within this time it is necessary to use `sniffle` in applications that require faster client-server communications, one obvious idea is to use a serial communications link. Visual Basic Professional Edition includes a control that makes it easy to have your favorite VB program talk to a modem. It should be possible to change the client-server communication channel of `sniffle` to a serial line without altering much of the system's core.

A more ambitious plan would be to create a system that permits real-time interaction between the remote controller and the I/O board.

6 Acknowledgements

This project would not have been possible without the generous assistance of a number of individuals and organizations.

Jay Lieske and Kirk Alexander of Princeton University's Computing and Information Technology department offered much useful advice during the planning stages of the project.

Calvin Diller of MetraTek Corporation provided crucial insights into serial communications and was generally extraordinarily helpful.

Bob Wojciechowski '97 showed me how to control I/O boards.

Bill Vinje of Princeton's Physics department devoted hours to an important late-night bug fix.

George Miller explained Tektronix digital oscilloscopes to me.

Bob Denny made the project possible by writing a robust, full-featured, well-documented httpd server and giving it away for free.

The customer support departments of Airpax Industries, Analog Devices, and Tektronix Corporation were particularly helpful.

Finally, Professor Michael Littman of Princeton's department of Mechanical and Aerospace Engineering provided all of the material resources behind the project, invaluable guidance, and a great deal of his time. He taught me how much can be accomplished with a positive attitude and a low budget.

A Project Log

At the outset of the `sniffle` project, I began to record each day's work in a log file. This file, listed below, provides a fairly complete account of the evolution of the project. It also contains lots of information about product vendors, "gotchas" that I encountered during the development process, and anything I considered worth writing down.

```
Log of activity on remote control via Web project with Prof. Littman.

Author:    Terence Kelly  <tpkelly@phoenix.princeton.edu>
RCS File:  $Source: /u/tpkelly/littman/RCS/log,v $
   $Revision: 1.9 $

6/6/95 Meeting with Jay Lieske and Kirk Alexander in ICGL
They recommend hardware encoding of MPEG.  Mention IRIS "InPerson"
software for SGI machines.  Names of CIT people who have clue:
Kevin Perry (Mathematica instructor), Dave Harrington (CU-SeeMe
experience).  Names of other people with clue:  Ken Stieglitz,
```

Bede Liu (image processing professor).

Littman discuss funding with Diane Balestri.

Suggest net search of "CU-SeeMe", "Live Video", "MPEG". See
Netscape page for server push/client pull.

Basic idea that emerged was to have two parallel systems solve
different parts of the problem: Use Web to transmit initial
conditions from client to lab-based server and to start
experiment. Meanwhile a separate system handles video feedback
from lab to client end using perhaps CU-SeeMe.

6/7/95 Met with Littman in lab. He approves of idea of two parallel
systems for instrument operation and video feedback. I said
I'd perform appropriate research.

6/8/95 Wrote Ted Nadeau asking for MPEG etc. advice
Faxed and snail-mailed to many many companies from magazine of
Kirk Alexander requesting MPEG hardware info etc.

Set up mail-logging scheme. Mail littman instead of mlittman
to log mail to ~tpkelly/littman/mbox.littman file.

Brief bobw on HTML and LaTeX

Surf net looking for httpd server info. Looks like Netscape
requires Windows NT. No mention of Linux among flavors of
Unix compatible with Netscape server.

NCSA mentions Intel-based PC at URL
http://hoohoo.ncsa.uiuc.edu/docs/setup/PreCompiled.html

Windows and Windows NT httpd at URL
http://www.yahoo.com/Computers/World_Wide_Web/HTTP/Servers/
  Windows___Windows_NT_HTTPD/

Cool Windows (not NT or 95, just Windows) server URL:
http://www.city.net/win-httpd/

Wrote to rdenny@netcom.com about above server, asking minimum
hardware requirements.

6/9 Got reply from rdenny, who says "A 386 should be fine, as long
as it has 4MB or more of memory."

CGI-DOS, helps you use Perl with Win-httpd, URL:
http://www.achilles.net/~john/cgi-dos/

Download Bob Denny's Windows httpd .zip file. Note the
directions on installation (we need recent PKUNZIP):

```
"Create a directory c:\httpd (don't be creative yet!),
and unzip the kit, using the PKUNZIP "-d" option to
preserve directories. Then open the file
C:\httpd\htdocs\index.html with your browser and read
it.  Once you have gone through the checklist, start
the server and access it as http://1.2.3.4/ where
1.2.3.4 is your PC's IP address. Try out the demo
page. If the server exits immediately, look in
c:\httpd\httpd.log for the reason."

Found CU-SeeMe page:  http://cu-seeme.cornell.edu/

Sunday June 11 Dartmouth will broadcast a CU-SeeMe video of
Bill Clinton's commencement speech:
HTTP://picard.dartmouth.edu/~oly/BillVision.html
There's a reflector at U-Penn, IP address 130.91.72.36,
according to Dartmouth docs.

CU-SeeMe Windows system requirements:
http://cu-seeme.cornell.edu/supportedwin.html
(looks like 386DX will suffice, but cheapest video board
will cost around $350)

Remote control on the Internet:
http://www.indstate.edu/msattler/sci-tech/comp/
misc/remote-control.html

CU-SeeMe User's Guide:
http://www.indstate.edu/msattler/sci-tech/comp/CU-SeeMe/how-to.html

6/10 Install CU-SeeMe on OPR PC in anticipation of Clinton Speech.
Connected to U-Penn reflector and caught glimpse of video broadcast.
IT WORKS!

MBONE FAQ at AT&T:  http://www.research.att.com/mbone-faq.html

6/11 Watched Clinton speech at Dartmouth.  Very disappointing.
At best got 1 frame per second.  Average was around 0.5 fps.
Worst was 0.1 fps.  Don't know source of bottleneck.

6/14 Talked to Littman.  He found interesting new Web page on remote
controlled CU-SeeMe camera:
http://www.hiof.no/ludvigsen/webdoc/pt/pt_dummy.html

He said I should start to configure httpd server on 386 on
cart, not to touch Robert's 386 unless other one is short of
disk space.  Our 386s have microchannel architecture, not
EISA.  The one on the cart has a phone-jack type ethernet
connector, Xoftware and Novell Winsock (?).  It's a PS/2
Model 80.
```

Phone numbers: G-101, where "George" often is: 5567
Littman's office: 5198

FTP'd Windows HTTPD and pkunzip.exe to C:\httpd\ on 386 on metal cart. That machine is a nightmare. Slow, poorly connected, software not configured properly (Xoftware is particularly uncooperative). Tried to pkunzip -d per the instructions but ran out of disk space. May have to do this on Robert's PC.

6/18 On 6/17 or thereabouts installed WinHTTPD on D:\ partition of 386 machine on cart in G-101. Configuration was a hassle because the httpd assumes you're installing on C:\, not D:\. But it works and successfully served a "hello world" document for over 24 hours. The URL is http://douglas.princeton.edu/.

6/19 Installed Netscape on two more PCs in G-101 (previously installed it on 386 on cart). Installed VBRUN300.DLL in C:\WINDOWS\SYSTEM on 386 on cart. Tested server: all features other than imagemap now seem to work properly. CGI-DOS and Visual Basic CGI stuff and Windows CGI stuff all seem to work. Had to edit PIFs for some DOS stuff to make it all work and make error messages go away. Server is still up and running as of 10:53 a.m.

6/24 Got "hello world" program to compile on Robert's PC and emit digital output to DVM via RTI-205 board and STB-50 connector cable. Program toggled voltage across outputs between 5.04 VDC and 0.033 VDC.

Problem 1: Could not compile a hello program on PC on metal cart where server is because compiler complained about DECL.TPU unit, even though I copied it directly from RTI205 directory, and then from Robert's C:\TP directory, where it worked. I think the problem may be that metal cart PC has later version of Turbo Pascal (a downgrade).

Was able to compile a HELLO2.EXE on Robert's PC and then run it on my PC, but it didn't produce the proper range of output voltages: output voltage oscillated reliably between 4.23 VDC and 4.37 VDC at correct times. Problem with STB-50? RTI-205?

7/6 Met Littman at noon and discussed plan for future. Here is current plan:

1. Client submits amplitude parameter to server, which passes parameter to batch file or one program therein.

2. O-scope begins to read & record data.

3. Pascal program gets amplitude parameter passed by client, outputs that voltage.

4. Everybody waits for a while.

5. Data is read from O-scope to file on server, then returned to client in native .WFM format.

6. Data is displayed at client using 222UTIL\VIEW.EXE

Sal Desiano gave me numbers for Analog Devices company (makers of RTI-205 board):

Main 617-461-4111
Regional 215-643-7790
Tech. Support 800-262-5643
Sales 617-461-4194

At noon demo'd to Littman activation of "Hello, World" program on server PC from Netscape client (the program output an analog voltage via the RTI-205).

In evening played with Tektronix 222 O-scope. Idea: use Tektronix utility VIEW.EXE (in C:\222UTIL) to view data captured by scope at client end: server transmits a .WFM ("waveform") file to client with new MIME type e.g. "application/waveform", and client spawns local copy of VIEW.EXE to view it.

7/8 Yesterday and today tested idea of using VIEW.EXE as Netscape helper application -- it works from my PC at OPR. It causes Netscape to issue a "floating point error: stack under" message on G-101 PS/2, but this is obviously not a problem with the basic idea.

Bill Vinje told me about "Lab View" -- a GUI for lab equipment interfaced to PCs, and about a standard protocol for communications beteween lab instruments and PCs called ??? (I forget) which allows a PC equipped with a special board to talk to a daisy-chain/bus of o-scopes, etc.. Must ask Littman about this.

Must call Tektronix and ask them for latest PC, Mac and X versions of VIEW utility.

7/9 Got following scheme to work well with RC circuit that operates on human time scales (using roughly 10 meg resistor and .22 uf cap):

Client submits voltage, delay1, delay2. Server activates Pascal program that drops output voltage to zero, waits delay1, raises output to voltage specified by user, waits delay2 sec., drops voltage to zero.

Can't get batch file that invokes above Pascal program to follow it with a GETWFM.EXE call -- resulting waveform file is empty. Possible solution: have Pascal program talk to oscilloscope to get waveform, rather than GETWFM.EXE.

7/10 Met with Littman in G-101 to demo above.

7/11 Looked at Tektronix ftp site ftp://ftp.tek.com/mbd/support where lots of utilities for 222 scope live. Unfortunately none of them are newer than the ones we have.

We should perhaps use TOADIF utility to convert .WFM files to .ADF format at server end, and view with WAVIEW at client end. May be difficult in any case to get client-spawned viewer to display two scope traces easily.

Problem: disk space is running low in lab.

7/17 Lately have had problems with RS-232 interface to Tektronix 222. Want to write little program to send command-line arguments to scope and write whatever comes back to standard output. This program CAN send strings like "BUT 1" to scope and scope usually responds appropriately. But a run-time hardware error (usually code 162) occurs when attempt to read from scope.

Called Tektronix customer support at 1-800-835-9433 x2400 (got the number from WWW page), spoke with Dave Frazel at x3028. He said problem might be our firmware version: we have version 2.16 whereas latest version is 3.8. He said he'd have one of his software people call me later today. The Tektronix customer support line is open 6:00 am to 5:00 pm Pacific Time. Tektronix customer support e-mail: tm_app_sup@tek.com.

Spoke with Ed Lingell about problem with 222. He wants to know what the error code means. He said Tektronix doesn't have source to 222UTIL stuff. Fellow who wrote it left company.

Later spoke with Tektronix Dave Nelson, who suggested that a "handshake error" might be responsible -- handshaking is set on the RS-232 board on the PC, in part using the DOS mode command. The settings that Nelson uses with Windows terminal are: CRLF, No parity, 8 data bits, 1 stop bit, no parity check. Nelson also gave me the number of Cal Diller, who wrote the 222UTIL stuff, and is now working for "Metro Tek": 503-640-4906.

Firmware upgrade: Washington DC service center (800-962-9738) says they don't handle 222 scopes. Referred me to Beaverton OR factory service (800-835-9433 x2400). Also gave me Ship To address:

Tektronix, Inc.
Attn: Factory Service, M/S 78-576

Regional Distribution Center
Beaverton OR 97077

Still waiting for pricing info on firmware upgrade from Tektronix.

Spoke with Barry Long of Analog Devices, requested info on all RTI-205-like products. He said recent versions of drivers are available from BBS: 617-461-4361, 8 data bits, 1 stop bit, no parity, 1200-9600 baud. Barry Long is an application engineer, his number is 617-461-3090. Application support is 617-461-4111.

Spoke with Cal Diller, author of 222UTIL stuff. His number at Metro Tek is given above, his e-mail is 71662.31@compuserve.com. He is VERY helpful. Said he'd mail me the source to 222UTIL stuff. Said in order to handle COM port well would have to write interrupt handler to deal with interrupt that occurs whenever the SINGLE BYTE of COM port buffer is written. 222UTIL things do NOT have an interrupt handler, just continuously read from port in tight loop. Said there is a Pascal command to get a character from the COM port. Said Visual Basic includes routines that do interrupt handling. Visual Basic is the way to go, says he. Recommends book, "The C Programmer's Guide to Serial Communications," by Campbell, published by SAMS. This book contains assembly code hardware interrupt handlers.

Can also write Windows DLLs in C/Pascal to use Windows interrupt handlers, and these can be called from Visual Basic.

Diller said he could give me source to API he wrote that lets you talk to instrument via hi-level commands, it runs as DLL and requires Visual Basic variable-length strings. Visual Basic costs $299 at Programmer's Paradise; Borland Delphi is $99, similar to Pascal.

Finally, Diller recommends National Instruments and their Lab View product: 6504 Bridge Point Parkway, Austin TX 78730-5039, 512-794-0100.

7/19 Installed Visual Basic on PC AT with 486 upgrade & 4 megs RAM. Have not tested yet. Wrote Cal Diller to ask if he knows about reading COM1 using Visual Basic. Documentation is silent about this.

7/21 Ordered MS Visual Basic Professional Edition thru CIT MDC, Nina Moyer 8-6798. Will deliver tomorrow (Saturday) to Littman home. Charged to account 150-1009. MDC orders from a company called "Ingram Micro" in or near Buffalo NY. Ingram Micro sales is 800-456-8000, customer support is 800-274-4800. They must receive an order by 3 p.m. to ship that day. MDC is 8-5566, open M-F 10:30-4:30 (at least at this time of year).

Egghead software in Lawrence was not helpful, could not get us product with educational discount. MDC price is $90, Egghead is $340.

Spoke with Wally Schwanke of Tektronix, 503-627-6630 regarding firmware upgrades. He said Tektronix now has 5-day tunaround time from time of receipt of scope to time of return shipment. He will investigate cost of upgrade and call back with info.

Nina Moyer of Microcomputer Distribution Center placed order of MS Visual Basic Pro, should be delivered to Littman home tomorrow.

7/23 Installed Visual Basic Pro, tested MSCOMM.VBX control. It works well.

7/24 Cal Diller's package arrived containing a diskette. Diller's post-it note on diskette says "I think the functions you want are in LIB\ASYNCXX.C". I copied the files on the diskette to C:\TPK\DILLER\ on my PC at work.

Also in Diller's package is a description of product by Diller's company, Metratek (503-693-1607): Waveform Manager Pro ($199). Will talk to 222 scopes, write GIF files. But can it be made to run in batch mode? Maybe using Visual Basic! Will ask Diller when I write to thank him.

7/25 Talked to three third-party vendors who make software for RTI series boards. DSP Development dude had little clue, said he'd send literature. HEM Data Corp sells boards, fully supports the 815 board, will send literature. LABTECH dude Brian Harrington was clueful & helpful, told me about LABTECH product Notebook (Pro) for data acquisition & talking to boards. Has some sort of macro feature called "batch runner" which sounds like exactly what we're looking for. Will send literature on products including educational discounts and bulk discounts. Phone numbers:

DSP Development 617-577-1133

HEM Data Corp. 810-559-5607

LABTECH 508-657-5400

I found out about these companies in the Analog Devices catalog.

Spoke with dude at DSP (toll free 800-424-3131 x400) who explained that their current products are more oriented toward data analysis rather than acquisition, therefore not appropriate for us.

7/27 Wally Schwanke of Tektronix (503-627-6630) said we could possibly get firmware upgrade for free but would require more details on how we got the scopes. Litman said Tektronix approached him around 1987 and offered to donate fancy equipment, he wrote up proposal, Tektronix gave us a dozen 222 scopes and one color printer. He was disappointed by what they gave us, wanted more. Gift came from "Signal and Measurement Division" or "Test and Measurement Division." Gift was in support of science education. Littman said could obtain copy of his proposal from Mary Lavin of MAE (melavin@pucc, 8-5127) or Anna Faiola. I wrote Mary Lavin at PUCC to ask for proposal.

GREAT IDEA: For dealing with all RTI board issues, simply write an INTERPRETER in Turbo Pascal or whatever that takes a text stream of commands and makes the RTI board do its thing based on those commands. E.g. if it gets a line that says "DOT 161" it tells the RTI board to output binary 161 on its eight digital output wires! Presto! We then have a LANGUAGE that can be transmitted over the Web that causes the RTI board to do ANYTHING it can possibly do!! Damn I'm good.

7/28 Got Visual Basic program with no forms whatsoever to run and respond; it returned something to the client. The thing to do is to have NO FORMS WHATSOEVER in your project. Use ONLY code modules, per R. Denny's suggestion in WinHTTPD documentation. Have a subroutine "Main" in one of the modules and designate this as the first thing to happen upon program execution.

Asked Littman about software & manuals for RTI-815. He authorized me to buy $120 worth of them from Analog Devices.

8/4 Purchased documentation, software and screw terminal board with ribbon cable for RTI-815. Tested board with EXER utility and DVM, found that analog outputs work. Modified Visual Basic sample program, tested it, found that it works. We get two analog outputs and possibly eight digital outputs. Have not yet been able to get digital outputs to work.

8/5 Digital outputs don't work because they operate off a SECOND set of output pins from a 34-pin J1 jumper in the middle of the RTI-815 board. Must purchase this ribbon because we need digital outputs.

One place to start looking for the GIF format spec is at
  http://www.cis.ohio-state.edu/hypertext/faq/usenet/graphics/
  fileformats-faq/part3/faq.html

Here's what that document has to say about the GIF format:

GIF is a data stream-oriented file format used to define the transmission protocol of LZW-encoded bitmap data. GIF images may be up to eight bits (256 colors) in depth and are always compressed. Despite the fact that GIF supports only 8-bits worth of colors, and the multimedia extensions introduced in the 89a release have not been widely utilized, GIF still remains a popular choice for storing lower resolution image data.

The GIF89a specification is available via many BBSs and on-line information services. You may also obtain the specification directly from CompuServe:

```
  CompuServe Incorporated
  Attn: Graphics Technology Department
  5000 Arlington Center Boulevard
  Columbus, OH 43220
  Voice: 614.457.8600, 800.848.8199
```

Lots of GIF utilities, plus a .ZIP file containing the format, are at http://ubu.hahnemann.edu/SimTel/gif.html.

The format spec itself is at
http://documents.cfar.umd.edu/imageproc/gif89a.doc

8/8 Ordered two proper 34-pin female J1 plugs (one with, one without polarizer) and ribbon cable from Master's Electronics in Raritan, 908-725-3533. Will be shipped today via first-class mail, arrive later in week, charged to my credit card. Cost will be under $15.

Fellow at Master's Electronics says that for LED hex readout of TTL lines, need 9368 chip (around $7 each) and 7-segment hex LED. He will see if he has pinout documentation on 9368 chip. Will take at least 2 weeks to order.

Littman hooked up his own hex LED displays for me, so no need to purchase. Tested digital outputs and they work well. Pinouts on hex LEDs are worth recording, because they're unusual:

```
                           --------
  Vcc   1  |  _   | 14   Vcc
 bit 1  2  | |_|  | 13  bit 2
 bit 0  3  | |_|  | 12  bit 3
4  |      | 11
  GND   5  |    | 10
        6  |    |  9
  GND   7  |______|  8   GND
```

8/10 Ordered from Free Software foundation Windows gnuplot diskette for Littman project; Texinfo manual for me; DJGPP CD-ROM and DOS utils diskettes for OPR. All should arrive via FedEx tomorrow.

Littman says re: Analog Devices bill to call purchasing, get PON, then have Analog Devices re-bill me. Call "Helen" in MAE if problems; she works in a.m. only. We didn't discuss FSF software (gnuplot for Windows) or $10 for J1 parts.

Littman and I added COM3 and COM4 ports to 486 PC; will use COM3 for Tektronix scope. Also replaced homebrew J1 plug with proper plug. Tested both scope (using 222UTILS) and RTI-815 digital outputs (using EXER), both seem to work.

At this point we don't know where our video server will live. Littman promises a video board before he leaves town on Saturday (Aug. 12 -- 2 days from now). We don't know what PC it will live on.

8/15 Interpreter basically works. Transitions on analog outputs require about 5 milliseconds.

8/17 Tested long "programs" containing digital outputs on my interpreter yesterday. When you submit a commandstring consisting of very many "DOT 0; DOT 1" commands, the time required for these commands gets longer as the commandstring "executes." The interval for which the low bit remains at 0 or 5 V begins at around 50 milliseconds, if I recall correctly, and increases up to around 250 milliseconds. Finally the VB program crashes with an "Out of string space" error message.

This led me to develop an un-locker to remove the lockfile in cases where the WEBRTI.EXE crashes before removing it at normal termination.

Today got an imagemap working. You see an 8-arrow'd Pan-n-Tilt controller form. Click on e.g. the down arrow and you get back an HTML doc saying "You hit DOWN." There's a bug in the R. Denny docs on the subject: you must enter "imagemap.exe" in the URL to which the ISMAP image is submitted, not just "imagemap" as the documents say.

8/19 Met with Littman for a while, demo'd the CU-SeeMe camera controller and WEBRTI program. He suggested that I add ability to read in digital and analog values from RTI-815. I told him I can do this. We went over stepper motors, he gave me a two-motor pan-and-tilt apparatus and documentation on how to use a chip to step the motors. He also showed me how to use an open-collector inverter to shield the RTI-815 from the deadly 12 V logic involved with the motor controllers.

Later in the evening I cleaned up the WEBRTI interpreter a lot. Now it gives correct and nice diagnostic feedback, correctly tracks "recursion" (i.e. "SOURCE" commands), etc..

Next order of business: Implement commands like "TEK," "LOOP," "DOTB," and input commands like "DINB," "DIN," "XAIN."

8/22 The fellow at CIT who co-ordinates the Fall Information Fair is Jon Edwards, jedwards@phoenix. Maybe Littman and I can demo there.

Late at night spent much time getting Tektronix scope to talk to

Visual basic. Crux of problem was proper settings of serial port memory addresses and IRQ numbers. The final working settings are therefore worth recording:

COM1 COM2

Address: 03F8 02F8
Baud rate: 2400 2400
Parity: N N
Data bits: 8 8
Stop bits: 1 1
IRQ: 4 3

It took an obscene amount of effort to get these settings right.

8/24 Spent a long time on the Tektronix routines. The behavior of the routines is probabilistic. When I send button commands to the scope to freeze the screen and stash waveforms CH1 and CH2 in RAM (with a view toward retrieving it shortly thereafter; the idea is to freeze both waveforms at the same time), sometimes things work correctly, other times the waveforms don't get stored. Might have to slow the routine down way slow to make it work, but early tests of this approach did not work well.

8/26 Thought of a great name for the project: SNIFFLE, the Simple Network Interface to the Full Functionality of Lab Equipment. (Thought of name "sniffle" days ago; it's the name of a friend's Saab sedan; only later realized acronym fit.)

I think that as of today the functionality of the Visual Basic program at the heart of SNIFFLE is complete and error-free. Now I must test the system thoroughly, work out all bugs, and document it.

Also on the to-do list: wire up the stepper motors and play with pan-n-tilt implementation.

8/28 Called AIRPAX company, manufacturer of stepper motors, regarding the motors we're using for pan-n-tilt stuff. They say our model K82231 is obsolete, but they'll FAX me complete specs on it. Last night I had trouble wiring up SAA 1027 controller chips because I lacked info on which resistor values were appropriate to our motors.

AIRPAX says their distributor is Sager Electronics, 800-724-37800 (yes, that phone number is correct -- just dial all of those digits and all will be well). Sager, like AIRPAX, won't give out prices on products until you're ready to buy. Weird.

9/5 Check to see if Visual Basic has an "eval" command. This would let us send VB programs from client to server and let us replace

```
our toy macro language with a true programming language containing
powerful commands.  I think I investigate this question months ago,
but Littman suggested the idea today and I said I'd check it out.

9/6 Got pan-n-tilt imagemap control panel to work.  Must be careful with
GET-method URLs in .map file:  if they're too long problems arise.
length limit seems ridiculously short.  Probably due to fact that
URL args are *last* thing passed to cgi program on command line.

Cleaned up source code by shortening all lines to under 75 characters
(except for one very long line that Visual Basic won't let us break).

9/7 Littman found a CU-SeeMe reflector on campus:  mongo.princeton.edu.
Littman says that X-terms can view the broadcast from our CU-SeeMe
server via MBONE something-or-other.  We haven't gotten this to work
yet.

Upon investigation we discovered a product called "nv" which
supposedly displays CU-SeeMe on an X-terminal.  It is installed in
/usr/princeton/bin/conferencing/, along with other related utilities
for internet teleconferencing (e.g. audio & whiteboards).  The nv
man pages are not installed by CIT, so it's best to get them from
the source:  ftp://parcftp.xerox.com/pub/net-research/nv*.  I've
put the man page in my ~tpkelly/man/man1 directory, along with a
.ps paper on the subject of nv.

Littman reports a bug in session logs of SNIFFLE:  Sometimes when
a barrage of commands is issued, or when two clients try to use
the server, bad things happen, e.g. wrong session log is returned.
I confirmed this problem when trying to demonstrate locking to
him.  Dunno what's up until I stare at the code.

Fixed bug in session logs later at night.  Now every time the
WEBRTI program executes a different log is generated.  When too
many log files pile up in a special session log directory, the
client is warned about it and given an opportunity to delete
them with a single click.

9/14 Documentation complete.  This log to be included in my project
report as an appendix.
```

B Project Correspondence

Early on, I arranged for all electronic mail related to the sniffle project to be logged in a special mailbox file. I had originally intended to include it here, but as the file is over five thousand lines long as of September 14, 1995, so it is not included here. Contact me by e-mail if you really need to read it.

C Code Listing

All of the Visual Basic code underlying `sniffle` is listed below. By default, Visual Basic saves files like `.BAS` files in an unreadable binary format. It is wise to change this default behavior and save *all* files as ASCII text where possible, in order to make listings like this possible and in order to facilitate the all-important business of version control. The code consists of several modules, one form, and a makefile for each of three executables. The code is profusely commented and should be reasonably easy to read. To read it, begin by learning Visual Basic. Then study the low-level utility functions in modules `utility.bas`, `rti-815.bas` and `tek222.bas`. Finally, read subroutine `main` in `webrti.bas`. That's where execution of the `WEBRTI.EXE` program begins.

The `sniffle` project was the author's first exercise in Visual Basic, and not surprisingly this code suffers from a number of deficiencies. An experienced programmer will be able to trace my climb up the learning curve by reading the code. One of the biggest stylistic problems with the code is that it does not adhere to a consistent naming convention. In a project of this size variables etc. ought to be named according to some well-defined convention. Unfortunately, toward the close of the project I didn't have time to worry about style points. :)

Visual Basic does not have a line-continuation command, so all commands must be written on a single (sometimes very long) line. In the listings below, lines that end in a backslash are actually long lines that have been broken solely for the purpose of this listing. *If you want to use this code you must first restore these broken lines to their original condition by removing the line breaks and backslashes!*

htmlutil.bas

```
'
'  Author:  Terence Kelly
'  Date:    8 September 1995
'
'  This module contains utilities to do HTML things.
'  It is intended for use in conjunction with my Web
'  work with Professor Littman.
'
'         1         2         3         4         5         6         7
'234567890123456789012345678901234567890123456789012345678901234567890
'

Function HTML_document (ByVal title$, ByVal body$) As String
    '
    '  Return an HTML document with the given contents in the
    '  title and body sections.
    '

    R$ = "Content-type: text/html" & CRLF() & CRLF() & "<html>"
    R$ = R$ & CRLF() & HTMLHead(title$) & CRLF() & "<body>" & CRLF()
    R$ = R$ & body$ & CRLF() & "</body>" & CRLF() & "</html>" & CRLF()

    HTML_document = R$

End Function

Function HTML_H1 (ByVal H$) As String
    '
```

```
    '  Return a string containing the argument within H1 tags
    '

    HTML_H1 = "<h1>" & H$ & "</h1>"

End Function

Function HTML_H2 (ByVal H$) As String
    '
    '  Return a string containing the argument within H2 tags
    '

    HTML_H2 = "<h2>" & H & "</h2>"

End Function

Function HTML_HR () As String
    '
    '  Output a <hr> tag
    '

    HTML_HR = CRLF() & "<hr>" & CRLF()

End Function

Function HTML_href (ByVal URL$, ByVal Text$) As String
    '
    '  Return an HTML hypertext link to the given URL displaying
    '  the given text.
    '

     R$ = "<a href=" & Chr$(34) & URL$ & Chr$(34) & ">"
     R$ = R$ & Text$ & "</a>"

    HTML_href = R$

End Function

Function HTML_pre (ByVal P$) As String
    '
    '  Output the given string within <pre> tags
    '

    HTML_pre = "<pre>" & CRLF() & P & CRLF() & "</pre>" & CRLF()

End Function

Function HTMLHead (ByVal title$) As String
    '
    '  Output an HTML header containing the given Title
    '
```

```
    R$ = "<head>" & CRLF() & "<title>" & title & "</title>"
    R$ = R$ & CRLF() & "</head>" & CRLF()

    HTMLHead = R$

End Function
```

httpdutl.bas

```
'
'  Author:  Terence Kelly (unless otherwise noted within individual
'                         functions)
'  Date:    30 August 1995
'
'  This module contains functions to parse strings from a WinHTTPD
'  server.
'

Function GetURLValue (ByVal VarName$, ByVal S$) As String
    '
    '  Given a "variable name" and a URL-encoded string, return the
    '  "value" corresponding to the name, or null if no "name=value"
    '  pair exists in the given string.
    '
    '  Note that GetURLValue("foo", "&&&foo=gollybar=asdf&bar=jkl&")
    '  returns "gollybar".  This is NOT a bug; the second argument
    '  is a malformed URL string, and this function does NOT check
    '  for errors like this.
    '

    R$ = ""

    ' Loop over each &-delimited field
    For I% = 1 To NumOccur(S$, "&") + 1
        CurrStr$ = Split(S$, I%, "&")
        If InStr(1, CurrStr$, VarName$ & "=") = 1 Then
            R$ = Split(CurrStr$, 2, "=")
            Exit For
        End If
    Next I%

    GetURLValue = R$

End Function

Function URLDecode (ByVal S$) As String
    '
    '  Given a URL-encoded string, un-escape it, i.e. convert escape
    '  sequences back into un-encoded form.  Note that I'm assuming
    '  that URL-encoding consists entirely of converting spaces to
```

```
    ' plus signs and certain other characters to sequences of the
    ' form "%HH" where HH is a pair of hex digits.  URL encoding
    ' might be more complicated than this.  I wasn't able to find
    ' any definitive answers to this question.
    '

    R$ = ""

    For I& = 1 To Len(S$)
        C$ = Mid$(S$, I&, 1)
        Select Case C$
        Case "%"
            ' Unescape hex code
            R$ = R$ & Chr$(HexToDecimal(Mid$(S$, I& + 1, 2)))
            I& = I& + 2  ' Increment index
        Case "+"
            R$ = R$ & " "
        Case Else
            R$ = R$ & C$
        End Select
    Next

    URLDecode = R$

End Function
```

purgelog.bas

```
'
' Author:  Terence Kelly
' Date:    8 September 1995
'
' This module serves exactly one purpose:  It produces an
' executable which deletes all .LOG files in the WEBRTI
' session log directory.
'

Global Const Pat$ = "C:\HTTPD\HTDOCS\WEBRTI\SESSION\*.LOG"

Sub Main ()
    '
    ' Delete all files that match pattern.  This is a good
    ' sample cgi-bin program.  It's about as simple as can
    ' be.
    '
    ' The error handler is important in order to prevent
    ' Visual Basic's annoying error message boxes from popping
    ' up on the server console screen, thus effectively crippling
    ' the server.  We just ignore errors.
    '
```

```
    On Error GoTo Main_Error

    C$ = CRLF()
    out$ = Split(Command$, 3, " ")
    before$ = Trim$(Str$(NumFiles(Pat$)))
    Kill (Pat$)
    after$ = Trim$(Str$(NumFiles(Pat$)))
    T$ = "PURGELOG report"
    B$ = HTML_H1("PURGELOG reports:") & C$
    B$ = B$ & "<p>" & C$ & "Number of files matching "
    B$ = B$ & Pat$ & " before deletion: "
    B$ = B$ & before$ & ".  After deletion: " & after$ & "."
    B$ = B$ & "</p>" & C$
    R$ = HTML_Document(T$, B$)
    Call ClobberFile(out$, R$)

Exit Sub

Main_Error:
    Resume Next

End Sub
```

purgelog.mak

```
PURGELOG.BAS
..\VB\UTILITY.BAS
..\VB\HTMLUTIL.BAS
ProjWinSize=0,-7,201,147
ProjWinShow=2
Title="PURGELOG"
ExeName="PURGELOG.EXE"
Path="C:\HTTPD\CGI-WIN"
```

rti-815.bas

```
'
'  Author:  Terence Kelly
'  Date:    31 August 1995
'
'  This module contains functions which communicate with the RTI-815
'  analog/digital I/O board.  From the user's manual, pinouts on board
'  are as follows:
'
'       Analog channel 0 = pin 39 (use pin 43 as ground)
'       Analog channel 1 = pin 40 (use pin 44 as ground)
'

'---------------------------------------------------------------------
' Following stolen from VBEX800.BAS (from Analog Devices Corp.):
```

```
Declare Sub Initialize Lib "LLWIN.DLL" (erstat%)
Declare Sub Ain815 Lib "LLWIN.DLL" (ByVal Lc%, ByVal Bd%, ByVal chan%, \
  ByVal gain%, erstat%)
Declare Sub Aot815 Lib "LLWIN.DLL" (ByVal Lc%, ByVal Bd%, ByVal chan%, \
  erstat%)
Declare Sub XAin Lib "LLWIN.DLL" (ByVal Lc%, rvalue!, erstat%)
Declare Sub XAot Lib "LLWIN.DLL" (ByVal Lc%, rvalue!, erstat%)

Global Const AotLc0% = 1, AotLc1% = 2

' Digital values added by T. Kelly:
Global Const DinLc% = 3, DotLc% = 4
' Logical channels 5 thru 12 reserved for DotB:
Global Const DotB_baseLc% = 5
' Logical channels 13 thru 20 reserved for DinB:
Global Const DinB_baseLc% = 13
' L.C.'s 21 thru 36 (at least!) reserved for XAin:
Global Const XAin_baseLc% = 21

Global Const board% = 1

' End of code from VBEX800.BAS
'--------------------------------------------------------------------

' Added by T. Kelly, following examples above:
Declare Sub Dot815 Lib "LLWIN.DLL" (ByVal Lc%, ByVal Bd%, erstat%)
Declare Sub Dot Lib "LLWIN.DLL" (ByVal Lc%, ByVal valueout%, erstat%)
Declare Sub Dotb815 Lib "LLWIN.DLL" (ByVal Lc%, ByVal Bd%, ByVal bit%, \
  erstat%)
Declare Sub Dotb Lib "LLWIN.DLL" (ByVal Lc%, ByVal valueout%, erstat%)
Declare Sub Din815 Lib "LLWIN.DLL" (ByVal Lc%, ByVal Bd%, erstat%)
Declare Function Din Lib "LLWIN.DLL" (ByVal Lc%, erstat%) As Integer

Function RTI_Ain (ByVal N$) As String
    '
    '  Assumes that appropriate initialization routines have
    '  been run.
    '
    '  Given an analog input channel number in the range 0-15,
    '  return the voltage on that wire or an error message.
    '

    On Error GoTo RTI_Ain_Error

    R$ = ""

    chan% = Val(N$)
    Call XAin(XAin_baseLc% + chan%, voltage!, Xerr%)
    If Xerr% <> 0 Then
        R$ = "Error " & Str$(Xerr%) & " (board) in function RTI_Ain: "
        R$ = R$ & RTI_ErrorMsg(Xerr%)
```

```
    Else
        R$ = "value = " & Format$(voltage!, "#0.000000")
        R$ = R$ & " at " & Format$(Now, "h:m:ss m/d/yy")
    End If

    RTI_Ain = R$

    Exit Function

RTI_Ain_Error:
    R$ = R$ & "Error " & Str$(Err) & " (VB) in RTI_Ain: "
    R$ = R$ & Error$(Err)
    Resume Next

End Function

Function RTI_Aot (ByVal CH$, ByVal V$) As String
    '
    '  Assumes that appropriate initialization routines have been run.
    '
    '  Given strings containing a physical channel and a voltage, use
    '  RTI -815 routines to place given voltage on given channel.
    '  Return string containing error messages.
    '

    On Error GoTo RTI_Aot_Error

    R$ = ""

    If CH$ = "0" Then
        Call XAot(AotLc0%, Val(V$), Xerr%)
    ElseIf CH$ = "1" Then
        Call XAot(AotLc1%, Val(V$), Xerr%)
    Else
        RTI_Aot = "Error in routine RTI_Aot: invalid channel argument"
        Exit Function
    End If

    If Xerr% <> 0 Then
        R$ = "Error " & Str$(Xerr%) & " (board) in routine RTI_Aot: "
        R$ = R$ & RTI_ErrorMsg(Xerr%)
    End If

    RTI_Aot = R$

    Exit Function

RTI_Aot_Error:
    R$ = R$ & "Error " & Str$(Err) & " in routine RTI_Aot: "
    R$ = R$ & Error$(Err)
    Resume Next
```

```
End Function

Function RTI_Din () As String
    '
    '  Returns a hex value in the range 0-FF read from the digital
    '  inputs of the RTI-815, or error messages.
    '

    On Error GoTo RTI_Din_Error

    R$ = ""

    val_read% = Din(DinLc%, Xerr%)
    If Xerr% <> 0 Then
        R$ = "Error " & Xerr% & " (board) in function RTI_Din: "
        R$ = R$ & RTI_ErrorMsg(Xerr%)
    Else
        R$ = "value = " & Hex$(val_read%) & " at "
        R$ = R$ & Format$(Now, "h:m:ss m/d/yy")
    End If

    RTI_Din = R$

    Exit Function

RTI_Din_Error:
    R$ = R$ & "Error " & Str$(Err) & " (VB) in routine RTI_Din: "
    R$ = R$ & Error$(Err)
    Resume Next
End Function

Function RTI_Dot (ByVal N$) As String
    '
    '  Assumes that appropriate initialization routines have been run.
    '  Given a string containing a decimal or hex number in range
    '  0-255, use RTI-815 to put this value on output wires.  String
    '  returned contains error messages.
    '
    '  Hex numbers must begin with "H" or "h" and be no longer than
    '  two digits long.  Decimal numbers must be in range 0-255.
    '

    On Error GoTo RTI_Dot_Error

    R$ = ""

    If Mid$(N$, 1, 1) = "H" Or Mid$(N$, 1, 1) = "h" Then
        Num% = HexToDecimal(Mid$(N$, 2))
    Else
        Num% = Val(N$)
```

```
    End If

    If Num% < 0 Or Num% > 255 Then
        R$ = "Error in routine RTI_Dot:  out-of-range parameter"
    Else
        Call Dot(DotLc%, Num%, Xerr%)
        If Xerr% <> 0 Then
            R$ = "Error " & Str$(Xerr%) & " (board) in routine RTI_Dot: "
            R$ = R$ & RTI_ErrorMsg(Xerr%)
        End If
    End If

    RTI_Dot = R$

    Exit Function

RTI_Dot_Error:
    R$ = R$ & "Error " & Str$(Err) & " (VB) in routine RTI_Dot: "
    R$ = R$ & Error$(Err)
    Resume Next
End Function

Function RTI_DotB (ByVal B$, ByVal V$) As String
    '
    '  Assumes that Initialize and DOTB815 have been called.
    '
    '  Given a bit position in the range 0-7 and a value of 0 or 1,
    '  write the given value to the given bit of the digital output.
    '  Return an error message if applicable.
    '
    On Error GoTo RTI_DotB_Error

    R$ = ""
    bit% = Val(B$)
    value% = Val(V$)

    If value% < 0 Or value% > 1 Or bit% < 0 Or bit% > 7 Then
        RTI_DotB = "Error in function RTI_DotB: invalid parameter"
        Exit Function
    End If

    Call Dotb(DotB_baseLc% + bit%, value%, Xerr%)
    If Xerr% <> 0 Then
        R$ = "Error " & Xerr% & " (board) in function RTI_DotB: "
        R$ = R$ & RTI_ErrorMsg(Xerr%)
    End If

    RTI_DotB = R$

    Exit Function
```

```
RTI_DotB_Error:
    R$ = R$ & "Error " & Str$(Err) & " (VB) in routine RTI_DotB: "
    R$ = R$ & Error$(Err)
    Resume Next

End Function

Function RTI_ErrorMsg (ByVal ErrNo%) As String
    '
    '  Given a number corresponding to an Analog Devices RTI-815 error
    '  code, return a string describing the error.  See p. A-1 of the
    '  Software Manual for a listing of these error codes.  I just
    '  typed in all the error messages. I carefully checked all of
    '  this for accuracy, but there's a chance I made a typo or other
    '  error.  When in doubt, rely on the error number and consult
    '  the Software Manual.
    '

    Select Case ErrNo%
    Case 0
        R$ = "No error detected. (For CHECK routine, operation is "
        R$ = R$ & "not complete.)"
    Case 101
        R$ = "Channel number is out of range."
    Case 102
        R$ = "Operation is not valid for board type."
    Case 103
        R$ = "Board number is out of range."
    Case 104
        R$ = "Configuration file not found."
    Case 105
        R$ = "Value argument is too large."
    Case 106
        R$ = "Value argument is too small."
    Case 107
        R$ = "Gain argument is invalid."
    Case 109
        R$ = "Conversion timeout error occurred."
    Case 110
        R$ = "For AIN/XAIN and AING/XAING operations, an analog "
        R$ = R$ & "input operation is in progress.  For COLLECT/XCOLL"
        R$ = R$ & " and SCAN/XSCAN operations, the selected DMA "
        R$ = R$ & "buffer is in use."
    Case 111
        R$ = "DMA is not available."
    Case 112
        R$ = "Scan or collect operation in progress is not for this"
        R$ = R$ & " logical channel."
    Case 114
        R$ = "Range argument is invalid."
    Case 115
```

```
        R$ = "Mult argument is invalid."
    Case 116
        R$ = "Count argument is invalid."
    Case 117
        R$ = "Terminal count was reached (not an error)."
    Case 118
        R$ = "Overrun error was detected. (Mult x range is too small.)"
    Case 119
        R$ = "Trigger source is invalid."
    Case 120
        R$ = "Digital channel (bit #) value is out of range (must be"
        R$ = R$ & " (0-7)."
    Case 121
        R$ = "Value argument is out of range (must be 1 or 0)."
    Case 122
        R$ = "Gate argument is out of specified range."
    Case 123
        R$ = "Gate channel is already in operation."
    Case 124
        R$ = "Specified channel is in operation."
    Case 125
        R$ = "Specified channel has not been initialized."
    Case 127
        R$ = "Specified channel is not presently in operation."
    Case 128
        R$ = "Operation is not finished."
    Case 129
        R$ = "Counter overflow was detected."
    Case 130
        R$ = "Edge argument is out of range."
    Case 131
        R$ = "Specified mode is invalid."
    Case 132
        R$ = "Oncycle argument is out of range."
    Case 133
        R$ = "Offcycle argument is out of range."
    Case 134
        R$ = "Period argument is out of range."
    Case 135
        R$ = "Counter value read is too large for integer variable "
        R$ = R$ & "(>32767)."
    Case 136
        R$ = "Timeout occurred waiting for gate channel to go high."
    Case 137
        R$ = "Operation can not be performed on this channel."
    Case 138
        R$ = "Digital I/O port number is out of range."
    Case 139
        R$ = "Error occurred reading configuration file."
    Case 140
        R$ = "AOT overrun error occurred."
```

```
    Case 141
        R$ = "DMA buffer was not allocated. (Use MLTDLOAD.)"
    Case 142
        R$ = "Library for the board was not included in the link "
        R$ = R$ & "statement."
    Case 143
        R$ = "Logical channel number is either too big or invalid."
    Case 144
        R$ = "Logical channel is in use."
    Case 145
        R$ = "Channel sequence list is full."
    Case 146
        R$ = "In Turbo Pascal, drivers were not loaded. (Run PASL800"
        R$ = R$ & " or USERPASL.)"
    Case 158
        R$ = "The logical channel buffer is full."
    Case 159
        R$ = "The logical channel buffer contains more than 40K bytes."
    Case 164
        R$ = "The SETOUT table is full."
    Case 165
        R$ = "The SETOUT table is invalid."
    Case 166
        R$ = "The board specified cannot be used as an ACTIO "
        R$ = R$ & "interrupt source."
    Case 172
        R$ = "The values you were attempting to convert were returned"
        R$ = R$ & " by an unsupported routine."
    Case 175
        R$ = "An invalid modtype was assigned to the channel."
    Case 176
        R$ = "Floating point positive overflow."
    Case 177
        R$ = "Floating point negative overflow."
    Case 184
        R$ = "The modtype for the CJC channel is not set."
    Case 192
        R$ = "DMA is not configured for this board."
    Case Else
        R$ = "Not a valid Analog Devices RTI-815 error number, "
        R$ = R$ & "according to T. Kelly's error message function."
        R$ = R$ & "  Refer to the Software Manual to be sure."
    End Select

    caveat$ = "  (Check this translation of error number in manual!)"
    RTI_ErrorMsg = R$ & caveat$

End Function

Function RTI_Initialize () As String
    '
```

```
' Initialize RTI-815 for digital and analog I/O.  Return string
' of error messages.
'

R$ = ""

Call Initialize(Xerr%)
If (Xerr% <> 0) Then
    R$ = "RTI_Initialize Error " & Str$(Xerr%) & " (board): "
    RTI_Initialize = R$ & RTI_ErrorMsg(Xerr%)
    Exit Function
End If

Call Dot815(DotLc%, board%, Xerr%)
If (Xerr% <> 0) Then
    R$ = R$ & "Dot815 Setup error " & Str$(Xerr%) & ": "
    R$ = R$ & RTI_ErrorMsg(Xerr%) & CRLF()
End If

Call Aot815(AotLc0%, board%, 0, Xerr%)
If (Xerr% <> 0) Then
    R$ = R$ & "Aot815 Setup error LC " & Str$(AotLc0%) & ": "
    R$ = R$ & Str$(Xerr%) & ": " & RTI_ErrorMsg(Xerr%) & CRLF()
End If

Call Aot815(AotLc1%, board%, 1, Xerr%)
If (Xerr% <> 0) Then
    R$ = R$ & "Aot815 Setup error LC " & Str$(AotLc1%) & ": "
    R$ = R$ & Str$(Xerr%) & ": " & RTI_ErrorMsg(Xerr%) & CRLF()
End If

For I% = 0 To 7
    Call Dotb815(DotB_baseLc% + I%, board%, I%, Xerr%)
    If (Xerr% <> 0) Then
        R$ = R$ & "Dotb815 Setup error at bit " & Str$(I%) & ": "
        R$ = R$ & Str$(Xerr%) & ": " & RTI_ErrorMsg(Xerr%) & CRLF()
    End If
Next

Call Din815(DinLc%, board%, Xerr%)
If (Xerr% <> 0) Then
    R$ = R$ & "Din815 error " & Str$(Xerr%) & ": "
    R$ = R$ & RTI_ErrorMsg(Xerr%) & CRLF()
End If

For I% = 0 To 15
    Call Ain815(XAin_baseLc% + I%, board%, I%, 1, Xerr%)
    If (Xerr% <> 0) Then
        R$ = R$ & "Ain815 Setup error at channel " & Str$(I%)
        R$ = R$ & ": " & Str$(Xerr%) & ": " & RTI_ErrorMsg(Xerr%)
        R$ = R$ & CRLF()
```

```
        End If
    Next

    RTI_Initialize = R$

End Function
```

tek222.bas

```
'
'  Author:  Terence Kelly
'  Date:    5 September 1995
'
'  This module contains functions for talking to the Tektronix 222
'  oscilloscope through a serial port.
'
'  On 8/22/95 Prof. Littman and I spent a great deal of time
'  configuring the serial port on the 486 used with the WEBRTI
'  project in order to enable it to speak to the Tektronix 222.
'  Here are the settings for COM1 and COM2 that we finally
'  settled on (we use COM1 for Tektronix communications):
'
'                                  COM1            COM2
'
'           Memory address:        03F8            02F8
'           Baud rate:             2400            2400
'           Parity:                N               N
'           Data bits:             8               8
'           Stop bits:             1               1
'           IRQ:                   4               3
'
'-----------------------------------------------------------------------------

Global Const Tek_COMPortNumber% = 1
Global Const Tek_COMPortSettings$ = "2400,N,8,1"

'  Keeps track of how many waveforms we've captured; used in
'  naming output files.
Global WaveformCount As Integer

Function Tek_AcqMode (ByVal N%) As String
    '
    '  Given a number in the range 0-3, return the corresponding
    '  acquisition mode string from table 5, page 13 of "222 RS-232
    '  Interface Guide."
    '

    R$ = ""

    If N% < 0 Or N% > 3 Then
        R$ = "Error in routine Tek_AcqMode: "
```

```
        Tek_AcqMode = R$ & Str$(N%) & " not valid parameter"
        Exit Function
    End If

    Select Case N%
    Case 0
        R$ = "NORM"
    Case 1
        R$ = "ENV"
    Case 2
        R$ = "AVG"
    Case 3
        R$ = "CENV"
    Case Else
        R$ = "Error in routine Tek_AcqMode: unclassified parameter"
    End Select

    Tek_AcqMode = R$

End Function

Function Tek_CPLG (ByVal N%) As String
    '
    '  Given a number in the range 0-3, return the corresponding
    '  CPLG string from tables 1 and 2, pp. 10-11 of "222 RS-232
    '  Interface Guide."
    '

    R$ = ""

    If N% < 0 Or N% > 3 Then
        R$ = "Error in routine Tek_CPLG: "
        Tek_CPLG = R$ & Str$(N%) & " is an invalid parameter"
        Exit Function
    End If

    Select Case N%
    Case 0
        R$ = "DC"
    Case 1
        R$ = "AC"
    Case 2
        R$ = "GND"
    Case 3
        R$ = "OFF"
    Case Else
        R$ = "Error in routine Tek_CPLG: unclassified parameter"
    End Select

    Tek_CPLG = R$
```

```
End Function

Function Tek_ErrorMsg (ByVal in$) As String
    '
    ' Given a string, search it for a Tektronix error message.  If
    ' an Error message is found in the input string, return the
    ' "long version" of the error message.  See p. 14 of the "222
    ' RS-232 Interfacing Guide" for details.  This function only
    ' handles status codes, not diagnostic error codes.  I've never
    ' had the scope give me a diagnostic error code, but it has
    ' often spit back status codes complaining about commands I've
    ' sent it.
    '

    R$ = ""
    S$ = Trim$(CondenseWhitespace(in$))
    I% = InStr(S$, "STA")
    If I% > 0 Then
        C$ = Mid$(S$, I%)
        Select Case Mid$(Split(C$, 2, " "), 1, 4)
        Case "0001"
            R$ = "Unrecognized command"
        Case "0002"
            R$ = "Unrecognized character"
        Case "0003"
            R$ = "Command is query only"
        Case "0004"
            R$ = "Command has no query"
        Case "0005"
            R$ = "Bad command argument"
        Case "0006"
            R$ = "Bad data"
        Case "0007"
            R$ = "Data is required"
        Case "0008"
            R$ = "Argument is required"
        Case "0009"
            R$ = "Communication task busy"
        Case "000A"
            R$ = "CURV command had bad checksum"
        Case "000B"
            R$ = "Bad task name for message"
        Case "FFFF"
            R$ = "User pressed escape"
        Case Else
            R$ = "WEBRTI function Tek_ErrorMsg does not recognize"
            R$ = R$ & " this status code"
        End Select
    End If

    If InStr(UCase$(S$), "ERR") Then
```

```
        If R$ <> "" Then R$ = R$ & ".  "
        R$ = R$ & "Diagnostic error code: see 222 RS-232 user "
        R$ = R$ & "interface guide"
    End If

    Tek_ErrorMsg = R$

End Function

Function Tek_FP (ByVal logfp$) As String
    '
    '  Get the front panel settings of the Tektronix scope and return
    '  them along with a human-readable decoding.
    '

    FP$ = Tek_send("3", "FP? " & logfp$)
    R$ = Trim$(Tek_InsertErrorMsgs(CondenseWhitespace(FP$)))

    If InStr(R$, "STA") Or InStr(R$, "ERR") Then
        Tek_FP = R$
    Else
        R$ = R$ & CRLF()
        Tek_FP = R$ & Tek_FPDecode(Mid$(FP$, InStr(FP$, ":") + 1, 10))
    End If

End Function

Function Tek_FPDecode (ByVal FP$) As String
    '
    '  Given a ten-character string representing the Tektronix 222
    '  response to the FP? command, decode this string into human-
    '  readable form according to tables 1 through 5, pp. 10 - 13
    '  of the "222 RS-232 Interface Guide."
    '

    If Len(FP$) <> 10 Then
        R$ = "Error in routine Tek_FPDecode: Len(parameter) != 10"
        Tek_FPDecode = R$
        Exit Function
    End If

    C$ = CRLF()
    R$ = ""

    ' channel 1
    Ch$ = Mid$(FP$, 1, 1)
    R$ = R$ & " CH1INV="
    R$ = R$ & UCase$(Format$(HexExtract(Ch$, 0, 1), "On/Off")) & C$
    R$ = R$ & " CH1VAR="
    R$ = R$ & UCase$(Format$(HexExtract(Ch$, 1, 1), "On/Off")) & C$
    R$ = R$ & "CH1CPLG=" & Tek_CPLG(HexExtract(Ch$, 2, 2)) & C$
```

```
    R$ = R$ & " CH1V/D=" & Tek_VoltsDiv(Mid$(FP$, 2, 1)) & C$

    ' channel 2
    Ch$ = Mid$(FP$, 3, 1)
    R$ = R$ & " CH2INV="
    R$ = R$ & UCase$(Format$(HexExtract(Ch$, 0, 1), "On/Off")) & C$
    R$ = R$ & " CH2VAR="
    R$ = R$ & UCase$(Format$(HexExtract(Ch$, 1, 1), "On/Off")) & C$
    R$ = R$ & "CH2CPLG=" & Tek_CPLG(HexExtract(Ch$, 2, 2)) & C$
    R$ = R$ & " CH2V/D=" & Tek_VoltsDiv(Mid$(FP$, 4, 1)) & C$

    ' Table 3: Sec/Div and Misc. settings
    Ch$ = Mid$(FP$, 5, 2)
    R$ = R$ & " RO-OFF="
    R$ = R$ & UCase$(Format$(HexExtract(Ch$, 0, 1), "On/Off")) & C$
    R$ = R$ & "     XY="
    R$ = R$ & UCase$(Format$(HexExtract(Ch$, 1, 1), "On/Off")) & C$
    R$ = R$ & " X10MAG="
    R$ = R$ & UCase$(Format$(HexExtract(Ch$, 2, 1), "On/Off")) & C$
    R$ = R$ & "SEC/DIV=" & Tek_SecDiv(HexExtract(Ch$, 3, 5)) & C$

    ' Table 4:  Trigger Position, Slope, Source and Mode Settings
    Ch$ = Mid$(FP$, 7, 2)
    R$ = R$ & "TRIGPOS=" & Tek_TrigPos(HexExtract(Ch$, 0, 2)) & C$
    R$ = R$ & "  SLOPE=" & Tek_TrigSlope(HexExtract(Ch$, 2, 1)) & C$
    R$ = R$ & "TRIGSRC=" & Tek_TrigSource(HexExtract(Ch$, 3, 2)) & C$
    R$ = R$ & "TRIGMOD=" & Tek_TrigMode(HexExtract(Ch$, 5, 3)) & C$

    ' Table 5:  Acquisition Mode and Misc. Settings
    Ch$ = Mid$(FP$, 9, 2)
    ' Decoding of Timeout is perverse; see table 5
    R$ = R$ & "    TIMEOUT="
    R$ = R$ & UCase$(Format$(HexExtract(Ch$, 0, 1) - 1, "Yes/No")) & C$
    R$ = R$ & "    CHANSEL="
    R$ = R$ & IIf(HexExtract(Ch$, 1, 1), "CH1", "CH2") & C$
    R$ = R$ & "RECALLEDWFM="
    R$ = R$ & UCase$(Format$(HexExtract(Ch$, 2, 1), "Yes/No")) & C$
    R$ = R$ & " VALIDSTORE="
    R$ = R$ & UCase$(Format$(HexExtract(Ch$, 3, 1), "Yes/No")) & C$
    R$ = R$ & "    ACQMODE=" & Tek_AcqMode(HexExtract(Ch$, 4, 2)) & C$
    R$ = R$ & "  STOREMODE="
    R$ = R$ & IIf(HexExtract(Ch$, 6, 1), "STORE", "NONSTORE") & C$
    R$ = R$ & "   AUTOTRIG="
    R$ = R$ & UCase$(Format$(HexExtract(Ch$, 7, 1), "Yes/No")) & C$

    Tek_FPDecode = R$

End Function

Function Tek_get () As String
    '
```

```
    '  Get waveforms CH1 and CH2, store them in files.  Use global
    '  integer to keep track of how many non-null waveform files
    '  we've succeeded in grabbing from the scope.  Return a string
    '  containing information on the number of bytes
    '
    '  We assume that the scope is in SSEQ trigger mode and that
    '
    '  See the Tektronix manual entitled "222 RS-232 Interface Guide"
    '  for details on the button numbering scheme, trigger modes, etc.
    '

    Const Delay$ = "7"     ' CURV? commands take a while.

    R$ = ""

    ' record the time that this routine is invoked
    CapTime$ = Format$(Now, "hh:mm:ss m/d/yy")
    ' grab the waveforms
    CH1$ = Tek_send(Delay$, "curv? ch1")
    CH2$ = Tek_send(Delay$, "curv? ch2")
    ' store nonzero length waveforms to disk
    If CH1$ <> "" Then
        WaveformCount = WaveformCount + 1
        Filename$ = RTIDir$ & "TEK_" & Format$(WaveformCount, "0000")
        Filename$ = Filename$ & ".WFM"
        Call ClobberFile(Filename$, CH1$)
    End If
    If CH2$ <> "" Then
        WaveformCount = WaveformCount + 1
        Filename$ = RTIDir$ & "TEK_" & Format$(WaveformCount, "0000")
        Filename$ = Filename$ & ".WFM"
        Call ClobberFile(Filename$, CH2$)
    End If
    R$ = " Bytes received: CH1=" & Trim$(Str$(Len(CH1$)))
    R$ = R$ & ", CH2=" & Trim$(Str$(Len(CH2$))) & " at " & CapTime$

    Tek_get = R$

End Function

Function Tek_initialize () As String
    '
    '  Initialize the MSCOMM control used to communicate with the
    '  222 oscilloscope. Return any error messages that occur during
    '  this process.
    '

    On Error GoTo Tek_Initialize_Error

    strErrors$ = ""
    Tek_initialize = ""
```

```
    frmWEBRTIcontrols.Hide   ' form containing MSCOMM control
    frmWEBRTIcontrols.Comm1.CommPort = Tek_COMPortNumber%
    frmWEBRTIcontrols.Comm1.settings = Tek_COMPortSettings$
    frmWEBRTIcontrols.Comm1.InputLen = 0
    frmWEBRTIcontrols.Comm1.InBufferSize = 4096
    frmWEBRTIcontrols.Comm1.PortOpen = True
    frmWEBRTIcontrols.Comm1.OutBufferCount = 0
    frmWEBRTIcontrols.Comm1.InBufferCount = 0
    frmWEBRTIcontrols.Comm1.Output = Chr$(13)
    Delay (1)
    strReply$ = Trim$(CondenseWhitespace(frmWEBRTIcontrols.Comm1.Input))
    frmWEBRTIcontrols.Comm1.PortOpen = False
    Tek_initialize = Tek_InsertErrorMsgs(strReply$) & strErrors$

Exit Function

Tek_Initialize_Error:
    If strErrors$ <> "" Then strErrors$ = strErrors$ & CRLF()
    T$ = strErrors$ & "Error " & Str$(Err)
    strErrors$ = T$ & " (VB) in function Tek_initialize: " & Error(Err)
    Resume Next

End Function

Function Tek_InsertErrorMsgs (ByVal in$) As String
    '
    '  Given a reply string from the Tektronix scope, insert a
    '  parenthesized error message within every semicolon-delimited
    '  individual reply containing a STA or ERR notice.
    '
    '  We assume that the last semicolon-delimited field of the
    '  string returned by the Tektronix scope is empty.
    '
    R$ = ""

    For I% = 1 To NumOccur(in$, ";")
        Msg$ = Split(in$, I%, ";")
        E$ = Tek_ErrorMsg(Msg$)
        If E$ <> "" Then
            R$ = R$ & Msg$ & " (" & E$ & "); "
        Else
            R$ = R$ & Msg$ & "; "
        End If
    Next

    Tek_InsertErrorMsgs = R$

End Function

Function Tek_SecDiv (ByVal N%) As String
    '
```

```
' Given an integer in the range 0 - 26 return a string
' containing the corresponding Seconds/Division setting listed
' in Table 3, p. 12 of the Tektronix "222 RS-232 Interface
' Guide " or an error message."
'
If N% < 0 Or N% > 26 Then
    R$ = "Error in routine Tek_SecDiv: " & Str$(N%)
    Tek_SecDiv = R$ & " not a valid parameter (out of range)"
    Exit Function
End If

Select Case N%
Case 0
    R$ = "50 nS"
Case 1
    R$ = "0.1 uS"
Case 2
    R$ = "0.2 uS"
Case 3
    R$ = "0.5 uS"
Case 4
    R$ = "1 uS"
Case 5
    R$ = "2 uS"
Case 6
    R$ = "5 uS"
Case 7
    R$ = "10 uS"
Case 8
    R$ = "20 uS"
Case 9
    R$ = "50 uS"
Case 10
    R$ = "0.1 mS"
Case 11
    R$ = "0.2 mS"
Case 12
    R$ = "0.5 mS"
Case 13
    R$ = "1 mS"
Case 14
    R$ = "2 mS"
Case 15
    R$ = "5 mS"
Case 16
    R$ = "10 mS"
Case 17
    R$ = "20 mS"
Case 18
    R$ = "50 mS"
Case 19
```

```
        R$ = "0.1 S"
    Case 20
        R$ = "0.2 S"
    Case 21
        R$ = "0.5 S"
    Case 22
        R$ = "1 S"
    Case 23
        R$ = "2 S"
    Case 24
        R$ = "5 S"
    Case 25
        R$ = "10 S"
    Case 26
        R$ = "20 S"
    Case Else
        R$ = "Error in routine Tek_SecDiv: unclassified parameter"
    End Select

    Tek_SecDiv = R$

End Function

Function Tek_send (ByVal Delay$, ByVal S$) As String
    '
    ' Send the given string to the Tektronix scope and return
    ' whatever the scope replies.  Wait Delay$ seconds before
    ' reading the scope's reply.  It is the user/programmer's
    ' responsibility to choose Delay$ wisely.
    '

    On Error GoTo Tek_send_Error

    D% = IIf(Val(Delay$) > 0, Val(Delay$), 1)
    frmWEBRTIcontrols.Comm1.PortOpen = True
    frmWEBRTIcontrols.Comm1.OutBufferCount = 0
    frmWEBRTIcontrols.Comm1.InBufferCount = 0
    frmWEBRTIcontrols.Comm1.Output = S$ & Chr$(13)
    Delay (D%)
    strReply$ = frmWEBRTIcontrols.Comm1.Input
    frmWEBRTIcontrols.Comm1.PortOpen = False

    Tek_send = strReply$

Exit Function

Tek_send_Error:
    R$ = "Error " & Str$(Err) & " (VB) in function Tek_send: "
    Tek_send = R$ & Error$(Err)
    Exit Function
```

```
End Function

Function Tek_TrigMode (ByVal N%) As String
    '
    '  Given a number in the range 0-3, return the corresponding
    '  trigger mode string from table 4, page 13 of "222 RS-232
    '  Interface Guide."
    '

    If N% < 0 Or N% > 3 Then
        R$ = "Error in routine Tek_TrigMode: " & Str$(N%)
        Tek_TrigMode = R$ & " not valid parameter (out of range)"
        Exit Function
    End If

    Select Case N%
    Case 0
        R$ = "NORM"
    Case 1
        R$ = "AUTOLVL"
    Case 2
        R$ = "AUTOBL"
    Case 3
        R$ = "SSEQ"
    Case Else
        R$ = "Error in routine Tek_TrigMode: unclassified parameter"
    End Select

    Tek_TrigMode = R$

End Function

Function Tek_TrigPos (ByVal N%) As String
    '
    '  Given an integer in the range 0-2, return the corresponding
    '  trigger position string from table 4, p. 13 of "222 RS-232
    '  Interface Guide."
    '
    If N% < 0 Or N% > 2 Then
        R$ = "Error in routine Tek_TrigPos: " & Str$(N%)
        Tek_TrigPos = R$ & " not valid parameter (out of range)"
        Exit Function
    End If

    Select Case N%
    Case 0
        R$ = "POST"
    Case 1
        R$ = "MID"
    Case 2
        R$ = "PRE"
```

```
    Case Else
        R$ = "Error in routine Tek_TrigPos: unclassified parameter"
    End Select

    Tek_TrigPos = R$

End Function

Function Tek_TrigSlope (ByVal N%) As String
    '
    '  Given a number in the range 0-1, return the corresponding
    '  trigger slope string from table 4, page 13 of "222 RS-232
    '  Interface Guide."
    '
    '  This function could really be replaced with an IIf statement.
    '

    If N% < 0 Or N% > 1 Then
        R$ = "Error in routine Tek_TrigSlope: " & Str$(N%)
        Tek_TrigSlope = R$ & " not valid parameter"
        Exit Function
    End If

    Select Case N%
    Case 0
        R$ = "NEG"
    Case 1
        R$ = "POS"
    Case Else
        R$ = "Error in routine Tek_TrigSlope: unclassified parameter"
    End Select

    Tek_TrigSlope = R$

End Function

Function Tek_TrigSource (ByVal N%) As String
    '
    '  Given a number in the range 0-3, return the corresponding
    '  trigger source string from table 4, page 13 of "222 RS-232
    '  Interface Guide."
    '

    If N% < 0 Or N% > 3 Then
        R$ = "Error in routine Tek_TrigSource: " & Str$(N%)
        Tek_TrigSource = R$ & " not valid parameter"
        Exit Function
    End If

    Select Case N%
    Case 0
```

```
        R$ = "VERT"
    Case 1
        R$ = "CH1"
    Case 2
        R$ = "CH2"
    Case 3
        R$ = "EXT"
    Case Else
        R$ = "Error in routine Tek_TrigSource: unclassified parameter"
    End Select

    Tek_TrigSource = R$

End Function

Function Tek_VoltsDiv (ByVal Ch$) As String
    '
    ' Given a hex character in the range 0 - C, return the
    ' corresponding volts/division setting from tables 1 and 2,
    ' pp. 10-11 of the "222 RS-232 Interface Guide"
    '

    N% = Val("&H" & Ch$)
    If Len(Ch$) <> 1 Or N% > 12 Or N% < 0 Then
        R$ = "Error in routine Tek_VoltsDiv: " & Ch$
        Tek_VoltsDiv = R$ & " is not a valid parameter (out of range)"
        Exit Function
    End If

    Select Case N%
    Case 0
        R$ = "5 mV"
    Case 1
        R$ = "10 mV"
    Case 2
        R$ = "20 mV"
    Case 3
        R$ = "50 mV"
    Case 4
        R$ = "0.1 V"
    Case 5
        R$ = "0.2 V"
    Case 6
        R$ = "0.5 V"
    Case 7
        R$ = "1 V"
    Case 8
        R$ = "2 V"
    Case 9
        R$ = "5 V"
    Case 10
```

```
        R$ = "10 V"
    Case 11
        R$ = "20 V"
    Case 12
        R$ = "50 V"
    Case Else
        R$ = "Error in routine Tek_VoltsDiv: unclassified parameter"
    End Select

    Tek_VoltsDiv = R$

End Function
```

unlock.bas

```
'
'  Author:  Terence Kelly
'  Date:    5 September 1995
'
'  This module serves exactly one purpose in life:  it helps you
'  "unlock" the WEBRTI program by deleting that program's lock file.
'  The name of the file is stored in a global variable.  A simple
'  message is returned to the Web client indicating whether the
'  unlock operation is successful or not.
'

Global Const strLockFile$ = "C:\HTTPD\HTDOCS\WEBRTI\WEBRTI.LOK"

Function Click_Here () As String
    '
    '  Return a string consisting of an HTML document containing a
    '  button to click to unlock the RTI board.
    '

    URL$ = "http://piper.princeton.edu/cgi-win/unlock.exe?unlock"
    Link$ = "<a href=" & Chr$(34) & URL$ & Chr$(34) & ">UNLOCK</a>"

    B$ = HTML_H1("Unlock the WEBRTI program") & CRLF()
    B$ = B$ & HTML_H2("Current lockfile:") & CRLF()
    B$ = B$ & HTML_pre(FileContents(strLockFile$)) & CRLF() & CRLF()
    B$ = B$ & "<p>" & CRLF() & "If you are <em>absolutely sure</em> "
    B$ = B$ & "that the " & CRLF()
    B$ = B$ & "WEBRTI program is really not in use, i.e. that the "
    B$ = B$ & "program is " & CRLF()
    B$ = B$ & "erroneously locked (due e.g. to a server or "
    B$ = B$ & "executable crash), " & CRLF()
    B$ = B$ & "then click on the UNLOCK link below." & CRLF()
    B$ = B$ & "</p>" & CRLF()
    B$ = B$ & HTML_H1(Link$) & CRLF()

    Click_Here = HTML_document("Unlock the WEBRTI program", B$)
```

```
End Function

Sub Main ()
    '
    '  Remove lock file.  Return a happy or sad message to the
    '  client.
    '

    strCmdLine$ = Command$
    strOutputFile$ = Split(strCmdLine$, 3, " ")
    strURLArgs$ = Split(strCmdLine$, 4, " ")

    If strURLArgs$ <> "unlock" Then
        strReturn = Click_Here()
    Else
        strReturn = Unlock_RTI()
    End If

    Filenumber% = FreeFile
    Open strOutputFile$ For Output As #Filenumber%
    Print #Filenumber%, strReturn
    Close #Filenumber%

End Sub

Function Unlock_RTI ()
    '
    '  Unlock the RTI board by deleting the lock file, return a
    '  diagnostic message to the user.
    '

    If Dir$(strLockFile$) <> "" Then
        Kill strLockFile
        Title$ = "Unlock succeeded"
        MSG$ = HTML_H1("Lock file removed")
    Else
        Title$ = "Unlock failed"
        MSG$ = HTML_H1("Lock file not found")
    End If

    Unlock_RTI = HTML_document(Title$, MSG$)

End Function
```

unlock.mak

```
UNLOCK.BAS
..\VB\UTILITY.BAS
..\VB\HTMLUTIL.BAS
ProjWinSize=96,8,208,124
```

```
ProjWinShow=2
Title="UNLOCK"
ExeName="UNLOCK.EXE"
Path="C:\HTTPD\CGI-WIN"
```

utility.bas

```
'
' Author:  Terence Kelly
' Date:    8 September 1995
'
' This module, currently named "UTILITY.BAS", will contain utility
' functions which should be useful in a wide variety of general
' applications.  Many of these functions will perform mundane string
' manipulations.  I'm accustomed to powerful string manipulation
' languages like AWK and PERL, and the functions here should provide
' some of the abilities we take for granted in those languages.
'
' Error handling is an issue which, as of 7/27/95, I have not
' considered in great detail.  I'll probably avoid the issue for
' as long as possible.  Error handling really should be added to
' all of these functions to make them bulletproof.
'

Sub AppendToFIle (ByVal Filename$, ByVal Message$)
    '
    ' Append given message string to given file without modification.
    ' This function should trap errors because it is possible for a
    ' file to be "locked" in DOS's brain-damaged way (if a file is
    ' "open", it cannot be deleted or written or even *read*!), in
    ' which case this would fail.
    '

    AppendToFile_Filenumber% = FreeFile
    Open Filename$ For Append As #AppendToFile_Filenumber%
    Print #AppendToFile_Filenumber%, Message$;
    Close #AppendToFile_Filenumber%

End Sub

Sub ClobberFile (ByVal Filename$, ByVal Message$)
    '
    ' Write given contents to specified file, overwriting whatever
    ' is there.  There should be an error handler here.
    '

    If InStr(Filename$, "*") Or InStr(Filename$, "?") Then
        Exit Sub                ' no wildcards allowed
    End If

    Filenumber% = FreeFile
```

```
    Open Filename$ For Output As #Filenumber%
    Print #Filenumber%, Message$;
    Close #Filenumber%

End Sub

Function CondenseWhitespace (ByVal S$) As String
    '
    ' Given a string, convert all consecutive sequences of spaces,
    ' tabs and newlines into a single space.  Does NOT strip leading
    ' & trailing whitespace.
    '

    LastWasSpace% = False

    For I& = 1 To Len(S$)
        C% = Asc(Mid$(S$, I&, 1))
        Select Case C%
        Case 8, 9, 10, 13, 32       ' backspace, tab, LF, CR, space
            If LastWasSpace% = False Then
                R$ = R$ & " "
            End If
            LastWasSpace% = True
        Case Else
            R$ = R$ & Chr$(C%)
            LastWasSpace% = False
        End Select
    Next

    CondenseWhitespace = R$

End Function

Function CRLF () As String
    '
    '  Return string consisting of carriage return plus linefeed.
    '  It is probably inefficient to use this function in most
    '  cases.  Better to use a global constant.
    '

    CRLF = Chr$(13) & Chr$(10)

End Function

Sub Delay (ByVal N%)
    '
    '  Wait a specified number of seconds (up to an artificial limit).
    '
    '  This is a crude way to wait an interval.  A more sophisticated
    '  method would be to use a VB timer control.
    '
```

```
    If N% > 120 Then
        Delay_interval% = 120
    Else
        Delay_interval% = N%
    End If
    BeginTime = Now
    Do While DateDiff("s", BeginTime, Now) < Delay_interval%
    Loop

End Sub

Function Filecontents (ByVal Filename$) As String
    '
    ' Given a filename, returns a string containing the entire
    ' contents of the given file.  Return empty string if file
    ' doesn't exist.
    '
    ' WARNING:  error checking here is minimal or nonexistent.
    ' This function is intended for use with text files only;
    ' I don't know what would happen if it were fed a binary file.
    ' Also, there is a limit on the length of a Visual Basic string.
    ' If this function is fed a file longer than that limit, results
    ' are undefined.
    '

    Dim Filenumber As Integer
    Dim strReturnValue As String

    On Error GoTo FileContents_Error

    Filecontents = ""   ' return empty string if file doesn't exist
    If Dir$(Filename$) = "" Then Exit Function
    strReturnValue = ""
    Filenumber = FreeFile
    Open Filename For Input As #Filenumber
    Do While Not EOF(Filenumber)
        strReturnValue = strReturnValue & Input$(1, #Filenumber)
    Loop
    Close Filenumber
    Filecontents = strReturnValue

Exit Function

FileContents_Error:
    R$ = "Error number " & Err & " (VB) in function "
    Filecontents = R$ & "FileContents: " & Error$
    Exit Function

End Function
```

```
Function HexExtract (ByVal H$, ByVal pos%, ByVal length%) As Long
    '
    ' Given a Hex string, a bit position, and a length, return the
    ' long value of the binary number of given length beginning at
    ' given bit position within given hex string.  For instance,
    ' given hex string "9C", position 3, length 3, return 7:
    ' 9CH = 10011100 binary, bit string of length 3 beginning at
    ' position 3 is 111 = 7.  First bit position is zero, at the
    ' LEFT of the string.
    '
    ' This function is mostly useful for decoding the stuff returned
    ' by the Tektronix scope FP? command.
    '

    If length% <= 0 Or pos% < 0 Or HexToDecimal(H$) <= 0 Then
        HexExtract = 0
        Exit Function
    End If

    firstchar% = (pos% \ 4) + 1
    lastchar% = ((pos% + length% - 1) \ 4) + 1
    T1& = HexToDecimal(Mid$(H$, firstchar%, lastchar% - firstchar% + 1))
    power% = 3 - ((pos% + length% - 1) Mod 4)
    T2& = Int(T1& \ (2 ^ power%))

    HexExtract = T2& Mod (2 ^ length%)

End Function

Function HexToDecimal (ByVal S$) As Long
    '
    ' This is really rather silly, since the Val function recognizes
    ' and convert hex numbers.  I wrote this before learning about
    ' this aspect of Val.  This function has somewhat more range than
    ' Val, however: according to the manual Val converts "&HFFFF" to
    ' -1, whereas my function shouldn't choke on it so quickly.
    '
    ' Use this only for long hex numbers; use Val for short ones.
    '
    ' Given a string containing a hexidecimal number, return the
    ' number as a decimal integer.  If number is invalid, return -1.
    '
    ' Exhaustively tested on numbers 0 thru 255 (decimal) on
    ' 9 August 1995
    '

    R& = 0
    PlaceValue& = 1
    For I& = Len(S) To 1 Step -1
        Select Case Mid(S$, I&, 1)
        Case "0"
```

```
            R& = R& + PlaceValue& * 0
        Case "1"
            R& = R& + PlaceValue& * 1
        Case "2"
            R& = R& + PlaceValue& * 2
        Case "3"
            R& = R& + PlaceValue& * 3
        Case "4"
            R& = R& + PlaceValue& * 4
        Case "5"
            R& = R& + PlaceValue& * 5
        Case "6"
            R& = R& + PlaceValue& * 6
        Case "7"
            R& = R& + PlaceValue& * 7
        Case "8"
            R& = R& + PlaceValue& * 8
        Case "9"
            R& = R& + PlaceValue& * 9
        Case "A", "a"
            R& = R& + PlaceValue& * 10
        Case "B", "b"
            R& = R& + PlaceValue& * 11
        Case "C", "c"
            R& = R& + PlaceValue& * 12
        Case "D", "d"
            R& = R& + PlaceValue& * 13
        Case "E", "e"
            R& = R& + PlaceValue& * 14
        Case "F", "f"
            R& = R& + PlaceValue& * 15
        Case Else
            GoTo HexToDecimal_Error
        End Select
        PlaceValue& = PlaceValue& * 16
    Next

    HexToDecimal = R&

Exit Function

HexToDecimal_Error:
    HexToDecimal = -1
Exit Function

End Function

Sub HexToDecimal_Test ()
    '
    ' Rename this "Main" to test Hex conversion function easily.
    '
```

```
    For I% = 1 To 255
        testv$ = Hex$(I%)
        rv& = HexToDecimal(testv$)
        msg$ = "I = " & I% & ": " & testv$ & " -> " & rv& & " -> "
        msg$ = msg$ & Hex$(rv&)
        MsgBox msg$
    Next

End Sub

Function NumFiles (ByVal pat$) As Long
    '
    '  Given a pathname (wildcard characters allowed) return the
    '  total number files with the given extension in the given
    '  directory.
    '

    count& = 0
    file$ = Dir$(pat$)
    While file$ <> ""
        count& = count& + 1
        file$ = Dir$
    Wend
    NumFiles = count&

End Function

Function NumOccur (ByVal S$, ByVal C$) As Long
    '
    ' Given a string S, return the number of C characters within it.
    '

    N& = 0
    For I& = 1 To Len(S$)
        If Mid$(S$, I&, 1) = C$ Then N& = N& + 1
    Next

    NumOccur = N&

End Function

Function Split (ByVal S$, ByVal N%, ByVal Delim$) As String
    '
    '  Given a string S, return the Nth field where fields are
    '  delimited by one or more Delim characters.  Return empty string
    '  if N = 0 or number of fields is less than N.  Inspired by the
    '  (far superior!) awk and perl "split" functions.  Delim must be
    '  a single character.  Passed many tests 7/27/95.  Updated
    '  without testing 8/8/95 to replace "As" in declaration with
    '  type-definition suffixes.  This shouldn't cause problems.
```

```
    '
    '  This should really be re-written to count fields the way
    '  awk does, at least in the case of non-whitespace delimiters.
    '

    Dim I As Integer
    Dim B As Integer    ' Beginning of argument to return
    Dim E As Integer    ' End of argument to return
    Const CASE_SENSITIVE = 0    ' Page 285 of _Language_Reference_

    Split = ""  ' Return zero-length string by default
    If Len(Delim) <> 1 Or N < 1 Or Len(S) < 1 Then Exit Function

    B = 0
    E = 0

    ' Set B to beginning of Nth field
    For I = 1 To N
        B = B + 1
        While Mid$(S, B, 1) = Delim And B <= Len(S)
            B = B + 1
        Wend
        If I < N Then
            While Mid$(S, B, 1) <> Delim And B <= Len(S)
                B = B + 1
            Wend
        End If
        If B > Len(S) Then Exit Function
    Next I
    ' Set E to end of Nth field
    E = B
    While Mid$(S, E, 1) <> Delim And E <= Len(S)
        E = E + 1
    Wend

    Split = Mid$(S, B, E - B)

End Function

Function StrSub (ByVal in$, ByVal old$, ByVal nu$) As String
    '
    '  Given a string, convert every instance of old to nu.
    '
    R$ = ""
    Temp$ = in$
    I% = 0
    Do While InStr(Temp$, old$)
        I% = InStr(Temp$, old$)
        R$ = R$ & Left$(Temp$, I% - 1) & nu$
        Temp$ = Mid$(Temp$, I% + Len(old$))
    Loop
```

```
    StrSub = R$ & Temp$

End Function
```

webrti.bas

```
'
'  Author:  Terence Kelly
'  Date:    8 September 1995
'
'  The .EXE program resulting from this module will be invoked by
'  an HTTPD server and will handle instructions from the Web
'  client.  Possible instructions include:  produce digital or
'  analog outputs via the RTI, communicate with the Tektronix 222,
'  and return data and messages to the client.
'
'  This module must never cause a window or icon to appear on
'  the server screen.  It must do its thing invisibly.
'  It must handle run-time errors intelligently and prevent the
'  appearance of windows.  See the documentation on Bob Denny's
'  WinHTTPD server, the RTI-815 board, and the Tektronix 222 scope
'  for details.
'

' Records basic usage info
Global Const RTILog$ = "C:\HTTPD\LOGS\WEBRTI.LOG"

' Server-end home of log files, etc.
Global Const RTIDir$ = "C:\HTTPD\HTDOCS\WEBRTI\"

' Client's view of RTIDir$
Global Const HREF$ = "http://piper.princeton.edu/webrti/"

' Pointers to individual scope dumps
Global Const TekLog$ = "TEK.HTM"

' This executable simply deletes all .LOG files from session log
' directory.
Global Const Purge$ = "http://piper.princeton.edu/cgi-win/purgelog.exe"

Global Const MAX_RECURSIONLEVEL% = 20
Global Const MAX_LOOPITERATIONS% = 100

' If the following file exists, WEBRTI will refuse to run.
Global Const Lockfile$ = "C:\HTTPD\HTDOCS\WEBRTI\WEBRTI.LOK"

'  Global variables used in error logging and execution tracking due
'  to "recursive" nature of execution flow:  ability to "SOURCE" a
'  server-end file containing a command string (which can itself
'  contain SOURCE commands) means we ought to track flow of execution
'  carefully.
```

```
Global SessionLog As String      ' Detailed info on an individual run

Global CommandNumber As Long     ' Never decremented.  Keeps track of
                                 ' non-null commands.

Global RecursionLevel As Integer ' Program submitted by client is at
                                 ' RecursionLevel zero. Commands in a
                                 ' file SOURCE'd by client program are
                                 ' at RecursionLevel 1; commands in
                                 ' file SOURCE'd by *that* file are
                                 ' at RecursionLevel 2, etc.

Global gstrLogString As String   ' See "Exec" routine.  For sake of
                                 ' efficiency, we accumulate log
                                 ' message in this string and write it
                                 ' to disk if it gets too big.  This
                                 ' lets us avoid a disk write for
                                 ' every call of "Exec" and also
                                 ' should prevent "out of string
                                 ' space" crashes.

' Following code is by R. Denny, author of WinHTTPD
'------------------------------------------------------------------
Declare Function GetPrivateProfileString Lib "Kernel" (ByVal lpSection As \
   String, ByVal lpKeyName As Any, ByVal lpDefault As String, ByVal \
   lpReturnedString As String, ByVal nSize As Integer, ByVal lpFileName \
   As String) As Integer

Type Tuple              ' Used for Accept: and "extra" headers
    Key As String       ' and for holding POST form key=value pairs
    value As String
End Type

Const MAX_ACCTYPE = 100     ' Added by T. Kelly

Global CGI_ProfileFile As String         ' Pathname of CGI Data File
Global CGI_AcceptTypes(MAX_ACCTYPE) As Tuple   ' Accept: types array
Global CGI_NumAcceptTypes As Integer ' No. of Accept: types in array

Const ENUM_BUF_SIZE = 8192     ' Size of key enumeration buffer

' End of code from R. Denny
'-------------------------------------------------------------------

Sub Debug_Main ()
    '
    '  Preliminary version of Main function.  Returns to client lots
    '  of info available to server.
    '
```

```
    Dim strCmdLine As String
    Dim strDataFile As String, strContentFile As String
    Dim strOutputFile As String, strURLArgs As String
    Dim Filenumber As Integer

    T$ = "BEGIN Debug_Main " & Format(Now, "h:m:ss m/d/yy")
    Call AppendToFIle(RTILog$, T$)
    strCmdLine = Command$

    strDataFile = Split(strCmdLine, 1, " ")    ' See WinHTTPD docs for
    strContentFile = Split(strCmdLine, 2, " ") ' explanation of these
    strOutputFile = Split(strCmdLine, 3, " ")  ' command-line arguments
    strURLArgs = Split(strCmdLine, 4, " ")

    CGI_ProfileFile = strDataFile       ' Bob Denny's global variable

    Filenumber = FreeFile
    Open strOutputFile For Output As Filenumber

    Print #Filenumber, "Content-type: text/html" & CRLF() & CRLF()

    '
    '  Send stuff to the client using Print #Filenumber commands
    '  here.  Be sure to send a valid HTML file.
    '

    Close #Filenumber

    T$ = "END Debug_Main " & Format(Now, "h:m:ss m/d/yy")
    Call AppendToFIle(RTILog$, T$)

End Sub

Sub Exec (ByVal S$)
    '
    '  Execute an individual command.  Global string is used to keep
    '  running track of commands executed so far and return values
    '  and error messages generated by those commands.  If this string
    '  gets too large, it is written to the output file and re-set to
    '  empty string.  This will hopefully avoid "out of string space"
    '  crashes because the string never gets very big.  It also avoids
    '  inefficient strategy of writing to disk on every call to this
    '  routine.
    '

    CommandNumber = CommandNumber + 1         ' increment global count

    ' dump global log message string to file and clear it if it's
    ' too big
    If Len(gstrLogString) > 4096 Then
        Call AppendToFIle(RTIDir$ & SessionLog, gstrLogString)
```

```
        gstrLogString = ""
    End If

    ' convert nulls back to semicolons, undoing the work of routine
    ' LoopHack
    CMD$ = StrSub(CondenseWhitespace(S$), Chr$(0), ";")

    LogMsg$ = CRLF() & String$(RecursionLevel * 3, " ") ' indent for
                                                       ' easy reading
    LogMsg$ = LogMsg$ & Trim$(Str$(RecursionLevel)) & "/"
    LogMsg$ = LogMsg$ & Trim$(Str$(CommandNumber))
    LogMsg$ = LogMsg$ & " : " & CMD$ & " : "
    gstrLogString = gstrLogString & LogMsg$

    RV$ = ""
    Select Case Split(CMD$, 1, " ")
        Case "TEK"
            SubCMD$ = Split(CMD$, 2, " ")
            Select Case SubCMD$
            Case "GET"
                RV$ = Tek_get()
            Case "SEND"
                T1$ = Tek_send(Split(CMD$, 3, " "), Loopstring(CMD$))
                T2$ = CondenseWhitespace(T1$)
                RV$ = Tek_InsertErrorMsgs(Trim$(T2$))
            Case "FP?"
                RV$ = Tek_FP(Split(CMD$, 3, " "))
            Case Else
                RV$ = "unrecognized subcommand to TEK"
            End Select
        Case "DOT"
            RV$ = RTI_Dot(Split(CMD$, 2, " "))
        Case "DOTB"
            RV$ = RTI_DotB(Split(CMD$, 2, " "), Split(CMD$, 3, " "))
        Case "DIN"
            RV$ = RTI_Din()
        '
        ' DINB GOES HERE -- NOT YET IMPLEMENTED
        '
        Case "XAOT"
            RV$ = RTI_Aot(Split(CMD$, 2, " "), Split(CMD$, 3, " "))
        Case "XAIN"
            RV$ = RTI_Ain(Split(CMD$, 2, " "))
        Case "SOURCE"
            FN$ = RTIDir$ & Split(CMD$, 2, " ") & ".RTI"
            FC$ = Filecontents(FN$)
            If RecursionLevel >= MAX_RECURSIONLEVEL% Then
                RV$ = "recursion level limit exceeded"
            ElseIf FC$ <> "" Then
                RecursionLevel = RecursionLevel + 1
                Call Interp(FC$)
```

```
                RecursionLevel = RecursionLevel - 1
            Else
                RV$ = "file empty or nonexistent: " & FN$
            End If
        Case "DELAY"
            Call Delay(Val(Split(CMD$, 2, " ")))
        Case "LOOP"
            If NumOccur(CMD$, "{") <> 1 Or NumOccur(CMD$, "}") <> 1 Then
                RV$ = "incorrect number of braces"
            ElseIf Val(Split(CMD$, 2, " ")) > MAX_LOOPITERATIONS% Then
                T$ = "loop iteration limit ("
                RV$ = T$ & Str$(MAX_LOOPITERATIONS%) & ") exceeded"
            ElseIf Val(Split(CMD$, 2, " ")) < 1 Then
                RV$ = "numeric argument invalid or less than one"
            Else
                For I% = 1 To Val(Split(CMD$, 2, " "))
                    Call Interp(Loopstring(CMD$))
                Next
            End If
        ' following case should handle more commands from RTI
        ' software manual
        Case "DINB", "OUTPUT"
            RV$ = "Routine Exec reports: command not yet implemented"
        Case Else
            RV$ = "Unrecognized command passed to routine Exec"
    End Select

    gstrLogString = gstrLogString & RV$

End Sub

'
' Following code from R. Denny's WinHTTPD distribution.
'
'---------------------------------------------------------------------------
'
'   GetAcceptTypes() - Create the array of accept type structs
'
' Enumerate the keys in the [Accept] section of the profile file,
' then get the value for each of the keys.
'---------------------------------------------------------------------------
Private Sub GetAcceptTypes ()
  Dim sList As String
  Dim I As Integer, J As Integer, l As Integer, n As Integer

  sList = GetProfile("Accept", "")        ' Get key list
  l = Len(sList)                          ' Length incl. trailing null
  I = 1                                   ' Start at 1st character
  n = 0                                   ' Index in array
  Do While ((I < l) And (n < MAX_ACCTYPE))' Safety stop here
    J = InStr(I, sList, Chr$(0))        ' J -> next null
```

```
    CGI_AcceptTypes(n).Key = Mid$(sList, I, J - I) ' Get Key, then value
    CGI_AcceptTypes(n).value = GetProfile("Accept", CGI_AcceptTypes(n).Key)
    I = J + 1                            ' Bump pointer
    n = n + 1                            ' Bump array index
  Loop
  CGI_NumAcceptTypes = n                  ' Fill in global count

End Sub

'
' Following code from R. Denny WinHTTPD distribution.
'
'---------------------------------------------------------------------------
'
'   GetProfile() - Get a value or enumerate keys in CGI_Data file
'
' Get a value given the section and key, or enumerate keys given the
' section name and "" for the key. If enumerating, the list of keys for
' the given section is returned as a null-separated string, with a
' double null at the end.
'
' VB handles this with flair! I couldn't believe my eyes when I tried
' this.
'---------------------------------------------------------------------------
Private Function GetProfile (sSection As String, sKey As String) As String
    Dim retLen As Integer
    Dim buf As String * ENUM_BUF_SIZE

    If sKey <> "" Then
        retLen = GetPrivateProfileString(sSection, sKey, "", buf, \
                 ENUM_BUF_SIZE, CGI_ProfileFile)
    Else
        retLen = GetPrivateProfileString(sSection, 0&, "", buf, \
                 ENUM_BUF_SIZE, CGI_ProfileFile)
    End If
    If retLen = 0 Then
        GetProfile = ""
    Else
        GetProfile = Left$(buf, retLen)
    End If

End Function

Sub Interp (ByVal S$)
    '
    '  Interpret a string of commands and "execute" them by passing
    '  individual non - Null commands to Exec routine.
    '
    '  Mismatched curly braces are such a scourge that we reject any
    '  commandstring unless curly braces match properly.
    '
```

```
    If NumOccur(S$, "{") <> NumOccur(S$, "}") Then
        T$ = "Error in routine Interp: mismatched curly "
        gstrLogString = T$ & "braces.  Exiting." & CRLF()
        Exit Sub
    End If

    ' We should really condense whitespace in the command below,
    ' rather than in Exec, for efficiency.
    NC$ = LoopHack(UCase$(NormalizeCommand(S$)))

    For I% = 1 To NumOccur(NC$, ";") + 1
        CMD$ = Trim$(Split(NC$, I%, ";"))
        If CMD$ <> "" Then
            Call Exec(CMD$)
        End If
    Next

End Sub

Function LoopHack (ByVal S$) As String
    '
    '  Given a string, return a string identical to the original
    '  except that between curly braces semicolons are converted to
    '  null characters.
    '
    '  We parse the command string by splitting on semicolons before
    '  dealing with loops. We want commands within a loop to be
    '  delimited by semicolons (to make life easier on users), but we
    '  don't want a loop command to be split on semicolons.  So we
    '  use a hack:  prior to parsing the commandstring, we replace
    '  semicolons with nulls.  Later on, within function Exec, we
    '  reverse the substitution.
    '
    '  Another use for this function is in dealing with multiple
    '  commands to send to the Tektronix scope.  Multiple commands
    '  are delimited by semicolons, and we want the user to be able
    '  to enter semicolons rather than some other delimiter. So we
    '  simply require the user to enclose multiple commands inside
    '  curly braces.
    '
    '  This function is simple and reliable, but a missing right
    '  curly brace can cause serious problems.  Therefore we check
    '  that braces match elsewhere, in routine Interp.
    '

    R$ = ""
    braces% = 0

    For I& = 1 To Len(S$)
        C$ = Mid$(S$, I&, 1)
```

```
        Select Case C$
        Case "{"
            braces% = braces% + 1
        Case "}"
            If braces% > 0 Then braces% = braces% - 1
        Case ";"
            If braces% > 0 Then C$ = Chr$(0)
        End Select
        R$ = R$ & C$
    Next

    LoopHack = R$

End Function

Function Loopstring (ByVal CMD$) As String
    '
    '  Given a string, extract from it the part between curly braces
    '  and return that part.  This function is used to turn the second
    '  argument of a LOOP command into a commandstring ready for the
    '  Interp routine.
    '
    '  This function is also used with the Tektronix SEND command to
    '  send multiple commands to the oscilloscope.
    '

    length& = InStr(CMD$, "}") - InStr(CMD$, "{") - 1

    If InStr(CMD$, "{") < 1 Then
        Loopstring = ""
    ElseIf length& > 0 Then
        T$ = Mid$(CMD$, InStr(CMD$, "{") + 1, length&)
        Loopstring = T$
    Else
        Loopstring = ""
    End If

End Function

Sub Main ()
    '
    '  Main subroutine.  Interfaces with WinHTTPD server via files.
    '  Processes client commands and passes them to RTI-815 and
    '  Tektronix 222 routines.  Obtains data from Tektronix 222 and
    '  returns it to the client.
    '

    ' First order of business: record startup in log file
    Starttime$ = Time$
    Startdatetime$ = Starttime$ & " " & Format$(Now, "m/d/yy")
    Timestamp$ = "BEGIN Main " & Startdatetime$
```

```
    Call AppendToFIle(RTILog$, Timestamp$ & CRLF())

''''''''''''''''
' DEBUG SECTION

' Exit Sub
''''''''''''''''

    ' initialize global variables
    CommandNumber = 0
    RecursionLevel = 0
    WaveformCount = 0
    gstrLogString = "BEGIN PROGRAM " & Startdatetime$ & CRLF()
    ' Session log file is named with an eight-digit number.  First
    ' two digits are day of month, next six digits are hour, minute
    ' and seconds of start time.
    T$ = Format$(Now, "dd") & StrSub(Starttime$, ":", "")
    SessionLog = "SESSION\" & T$ & ".LOG"

    ' get command-line arguments
    strCmdLine$ = Command$
    strDataFile$ = Split(strCmdLine$, 1, " ")    ' See WinHTTPD docs
    strContentFile$ = Split(strCmdLine$, 2, " ") ' for explanation of
    strOutputFile$ = Split(strCmdLine$, 3, " ")  ' command-line args
    strURLArgs$ = Split(strCmdLine$, 4, " ")
    CGI_ProfileFile = strDataFile$

    ' If RTI board is locked, return contents of lockfile and pointer
    ' to unlock utility, otherwise create a lockfile and proceed.
    If Dir$(Lockfile$) <> "" Then
        UnlockFile$ = "http://piper.princeton.edu/cgi-win/unlock.exe"
        URL$ = HTML_href(UnlockFile$, "Unlock WEBRTI")
        B$ = HTML_H1("File lock: ") & HTML_pre(Filecontents(Lockfile$))
        B$ = B$ & CRLF() & HTML_H2(URL$) & CRLF()
        Doc$ = HTML_Document("WEBRTI locked", B$)
        Call AppendToFIle(strOutputFile$, Doc$)
        Exit Sub
    Else
        T$ = Format$(Now, "h:m:ss m/d/yy") & " "
        T$ = T$ & GetProfile("CGI", "Remote Host") & " "
        T$ = T$ & GetProfile("CGI", "Remote Address") & CRLF()
        Call AppendToFIle(Lockfile$, T$)
    End If

    ' Remove Tektronix session log from previous run of program.
    ' It is necessary to check if it exists first to prevent a
    ' run-time error.
    If Dir$(RTIDir$ & TekLog$) <> "" Then
        Kill RTIDir$ & TekLog$
    End If
```

```
' initialize RTI board
gstrLogString = gstrLogString & "RTI Initialization: "
gstrLogString = gstrLogString & RTI_Initialize() & CRLF()

' initialize the COM port used with the oscilloscope
gstrLogString = gstrLogString & "COM Initialization: "
gstrLogString = gstrLogString & Tek_initialize() & CRLF()

' Look for value of "commandstring" in URL argument, then
' content-file
arg$ = GetURLValue("commandstring", strURLArgs$)
cont$ = GetURLValue("commandstring", Filecontents(strContentFile$))
strEncoded$ = IIf(arg$ <> "", arg$, cont$)
strDecoded$ = URLDecode(strEncoded$)
out$ = "encoded commandstring=" & strEncoded$ & CRLF()
Call AppendToFIle(RTILog$, out$)
out$ = "normalized commandstring=" & NormalizeCommand(strDecoded$)
out$ = out$ & CRLF()
Call AppendToFIle(RTILog$, out$)


''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
' ****************************************************** '
' *                                                    * '
' *  This is the part where all of the important stuff * '
' *  begins.  If you want to understand how this code  * '
' *  works, begin tracing execution here, at Interp(). * '
' *                                                    * '
' ****************************************************** '
''''''''''''''''''''''''''''''''''''''''''''''''''''''''''''
Call Interp(strDecoded$)


' write to session log files
T$ = CRLF() & CRLF() & "END PROGRAM "
T$ = T$ & Format$(Now, "h:m:ss m/d/yy") & CRLF()
Call AppendToFIle(RTIDir$ & SessionLog, gstrLogString & T$)
Call ClobberFile(RTIDir$ & TekLog$, TekPointersFile())

' if the data body received contains a value "rf" ("returnfile"),
' then return the contents of that file, else return a file of
' pointers to session logs & scope dumps
T$ = URLDecode(GetURLValue("rf", strURLArgs$))
If T$ = "" Then
    T$ = GetURLValue("rf", Filecontents(strContentFile$))
End If
RF$ = URLDecode(T$)
If RF$ <> "" Then
    Call AppendToFIle(strOutputFile$, Filecontents(RTIDir$ & RF$))
Else
    Call AppendToFIle(strOutputFile$, ReturnFile())
```

```
    End If

    Unload frmWEBRTIcontrols

    ' Unlock the WEBRTI program.  We first check that it's there
    ' to avoid a run-time error which would otherwise occur if someone
    ' forcibly unlocked the system during execution of this process.
    If Dir$(Lockfile$) <> "" Then Kill Lockfile$

    ' Last order of business: record termination in log file
    T$ = "END Main " & Format$(Now, "h:m:ss m/d/yy") & CRLF() & CRLF()
    Call AppendToFIle(RTILog$, T$)

End Sub

Function NormalizeCommand (ByVal CMD$) As String
    '
    '  Given a command string (already decoded -- NOT URL-encoded),
    '  normalize it by replacing CRLF with semicolons, stripping
    '  comments.  Note:  this method is buggy because a semicolon
    '  WITHIN a comment causes mayhem.
    '
    '  This should be re-written using my Strsub routine, in the
    '  UTILITY.BAS module.
    '

    R1$ = ""

    ' convert CRLF to semicolon, assuming that carriage returns and
    ' linefeeds always occur together (reasonable assumption on DOS
    ' system)
    For I& = 1 To Len(CMD$)
        C$ = Mid$(CMD$, I&, 1)
        Select Case C$
        Case Chr$(13)             ' strip carriage returns
            R1$ = R1$
        Case Chr$(10)             ' replace newline with semicolon
            R1$ = R1$ & ";"
        Case Else
            R1$ = R1$ & C$
        End Select
    Next

    ' strip out comments, defined as anything from a "#" to a
    ' semicolon
    R2$ = ""
    For I& = 1 To Len(R1$)
        C$ = Mid$(R1$, I&, 1)
        If C$ = "#" Then
            Do While C$ <> ";" And I& < Len(R1$)
                I& = I& + 1
```

```
                C$ = Mid$(R1$, I&, 1)
            Loop
        End If
        R2$ = R2$ & C$
    Next
    NormalizeCommand = R2$

End Function

Function ReturnFile () As String
    '
    '  Return a string consisting of an HTML file to be returned to
    '  the client.  This HTML file contains pointers to various other
    '  files containing information of interest to the client-end
    '  user.
    '

    DQ$ = Chr$(34)                          ' Double-quote character
    Title$ = "Feedback from WEBRTI.EXE"
    C$ = CRLF()
    SL$ = StrSub(SessionLog, "\", "/")  ' must use forward slashes
                                        ' in hyperlinks

    F$ = C$ & HTML_H1("Data files generated:") & C$ & C$
    F$ = F$ & "<dl>" & C$
    F$ = F$ & "   <dt><a href=" & DQ$ & HREF$ & SL$ & DQ$
    F$ = F$ & ">Session Log</a>" & C$
    F$ = F$ & "   <dd>A detailed record of every command executed by "
    F$ = F$ & "the WEBRTI.EXE" & C$
    F$ = F$ & "       interpreter, including error messages resulting"
    F$ = F$ & " from each command." & C$
    F$ = F$ & "       Values returned by input commands like DIN and "
    F$ = F$ & "XAIN are recorded" & C$
    F$ = F$ & "       here too." & C$
    F$ = F$ & "   <dt><a href=" & DQ$ & HREF$ & TekLog$ & DQ$
    F$ = F$ & ">Tektronix Dumps</a>" & C$
    F$ = F$ & "   <dd>Links to individual .WFM files containing "
    F$ = F$ & "oscilloscope traces" & C$
    F$ = F$ & "       from the Tektronix 222." & C$
    F$ = F$ & "</dl>" & C$ & C$

    '
    '  Return a warning message if there are lots of .LOG files in
    '  the session directory.  Each of these files can be arbitrarily
    '  large, so it's important that they be purged!
    '
    SD$ = RTIDir$ & "SESSION\"
    pat$ = SD$ & "*.LOG"
    NF& = NumFiles(pat$)
    If NF& > 20 Then
        F$ = F$ & "<hr>" & C$ & C$ & "<p>" & C$
```

```
        F$ = F$ & "<strong>NOTE:</strong> Session directory" & C$
        F$ = F$ & SD$ & C$ & " contains " & Trim$(Str$(NF&))
        F$ = F$ & " .LOG files.  Delete if no longer needed!" & C$
        F$ = F$ & "After you have viewed the current log you can" & C$
        T$ = "click here to purge session log directory."
        F$ = F$ & HTML_href(Purge$, T$)
        F$ = F$ & C$ & "</p>" & C$ & C$
    End If

    ReturnFile = HTML_Document(Title$, F$)

End Function

Function TekPointersFile () As String
    '
    ' Return an HTML file containing pointers to HTML files generated by
    ' the Tek_get routine.
    '

    Title$ = "Pointers to Oscilloscope Waveforms"
    Body$ = HTML_H1(Trim$(Str$(WaveformCount)) & " traces generated:")
    If WaveformCount > 0 Then
        Body$ = Body$ & CRLF() & "<ol>" & CRLF()
        For I% = 1 To WaveformCount
            Filename$ = "TEK_" & Format$(I%, "0000") & ".WFM"
            link$ = HTML_href(HREF$ & Filename$, Filename$)
            Body$ = Body$ & "<li>" & link$ & CRLF()
        Next
        Body$ = Body$ & "</ol>"
    Else
        Body$ = HTML_H1("No waveforms generated")
    End If

    ' yet another hack:  strip out the MIME header that function
    ' HTML_Document inserts into its output
    Bad$ = "Content-type: text/html" & CRLF() & CRLF()
    T$ = StrSub(HTML_Document(Title$, Body$), Bad$, "")
    TekPointersFile = T$

End Function
```

webrti.frm

```
VERSION 2.00
Begin Form frmWEBRTIcontrols
   BackColor       =   &H000000FF&
   BorderStyle     =   1  'Fixed Single
   Caption         =   "AAAARRRGGGHH!  This should NEVER be visible! \
        Nofity T. Kelly if you see this!"
   ClientHeight    =   2475
   ClientLeft      =   1635
```

```
ClientTop       =   3000
ClientWidth     =   9270
ControlBox      =   0   'False
Height          =   2880
Icon            =   WEBRTI.FRX:0000
Left            =   1575
LinkTopic       =   "Form1"
MaxButton       =   0   'False
MinButton       =   0   'False
ScaleHeight     =   2475
ScaleWidth      =   9270
Top             =   2655
Visible         =   0   'False
Width           =   9390
Begin MSComm Comm1
   InBufferSize    =   4096
   Interval        =   1000
   Left            =   120
   Settings        =   "2400,n,8,1"
   Top             =   120
End
Begin Label Label2
   Alignment       =   2  'Center
   AutoSize        =   -1  'True
   BackColor       =   &H0080FFFF&
   Caption         =   "This window should never be visible during the \
execution of the WEBRTI program.  See Bob \
Denny's WinHTTPD documetnation for details. \
(Basically, the back-end programs spawned by \
WinHTTPD must never cause a change in the \
appearance of the server console.)  This form \
is here only as a home for a few necessary \
controls."
   FontBold        =   -1  'True
   FontItalic      =   0   'False
   FontName        =   "MS Sans Serif"
   FontSize        =   9.75
   FontStrikethru  =   0   'False
   FontUnderline   =   0   'False
   Height          =   960
   Left            =   240
   TabIndex        =   1
   Top             =   1200
   Width           =   8775
   WordWrap        =   -1  'True
End
Begin Label Label1
   Alignment       =   2  'Center
   AutoSize        =   -1  'True
   BackColor       =   &H0000FFFF&
   Caption         =   "IF YOU CAN SEE THIS THEN SOMETHING IS \
```

```
   TERRIBLY WRONG!"
      FontBold        =   -1  'True
      FontItalic      =   0   'False
      FontName        =   "MS Sans Serif"
      FontSize        =   18
      FontStrikethru  =   0   'False
      FontUnderline   =   0   'False
      Height          =   975
      Left            =   720
      TabIndex        =   0
      Top             =   120
      Width           =   8175
      WordWrap        =   -1  'True
   End
End
```

webrti.mak

```
..\VB\HTMLUTIL.BAS
..\VB\UTILITY.BAS
WEBRTI.BAS
C:\WINDOWS\SYSTEM\MSCOMM.VBX
..\VB\RTI-815.BAS
..\VB\HTTPDUTL.BAS
..\VB\TEK222.BAS
WEBRTI.FRM
ProjWinSize=2,5,186,222
ProjWinShow=2
Title="WEBRTI"
ExeName="WEBRTI.EXE"
Path="C:\HTTPD\CGI-WIN"
```