[osmia rufus]

OSMIA

Title:	Open Source Medical Image Analysis
Acronym:	OSMIA
Date:	June 2003
Project Number:	IST-2001-34512
Deliverable:	D3.4 Specification of OSMIA Web Interface
Version:	1.0
Author(s)	Tony Lacey, *University of Manchester*
	Ian Poole, *Voxar Ltd.*

Contents

1 Introduction

The Web Analysis Server (WAS) was designed the aim to provide access to tina analysis functions using a framework which relied on the minimum of universally available technology. The current tinatool interface is unsophisticated by modern standards and therefore easy to duplicated (in the most part) using less aggressive GUI technologies than those now widely available. Indeed so limited are the demands of the tinatool interface that it was decided that the important functionality could be constructed using standard HTML mechanisms and rendered using a web browser. This provides some immediate benefits;

- HTML is well defined, robust, open standard and is supported (at least as a subset) by all the browser suppliers.
- A compliant browser is available for virtually every operating system.
- A graphical layer described using a human readable script-type approach is easy to modify and debug.
- Provides a flexible document oriented interface which naturally supports an educational approach to analysis.

Although the use of HTML as a GUI implies a network oriented motivation, this is not the case. The original intention was simply to run the server in the background, on the same machine as the browser. Internet technologies were selected to simplify the development process whilst meeting the main aim. Such technologies being readily available on a wide variety of platforms. However, the server is able to accept request from a browser running on any machine. Thus remote operation is achieved without any modifications. This has many potential uses;

- Allowing uses to have multiple on-going process running simultaneously.
- Monitoring of time expensive, long running processes.
- Assisting in remote collaborations; allowing developers to quickly demonstrate new techniques and interfaces.
- Centralising the analysis capability for multiple users in order to provide better hardware support and quality control; 'job' based submissions.

1.1 About this document

This document describes the aims, technologies and construction of the Web Analysis Server (WAS). The aim is to provide the reader with sufficient information that they will be able to install, run and use the system as well as improve, develop and extend its capabilities.

2 System Overview

The diagram of figure 1 provides an overview of the analysis server. The dotted line separates the client side of the connection (upper part) from the server side of the connection (bottom part). The client side is comprised of up to two components; a web browser (required) and a DICOM client (optional). The web browser is responsible for rendering the control interface and presenting the results. The DICOM client can be used to transfer image data to and from the server.

The server is built around a extendable open source web server known as *tclhttpd* (HTTP stands for Hyper-Text Transfer Protocol), a Tcl language scripted web server. Tcl is an interpreted script language and thus the tclhttpd server is stored in a series of non-compiled, human readable files. To run the web server requires a a Tcl shell known as a `tclsh`. However, in this case we extend the shell with tina commands as described in section 6. Executing the scripts using this shell provides the basis of the HTTP driven tinatool. The capabilities of this server are extended to cover the image analysis functionality provided by tinatool by adding new scripts to the tclhttpd custom folders, details of which are provided later.

The DICOM server is a standalone application which is linked to the analysis server via the scripting interface. One of the advantages of using the Tcl scripted HTTP server is that it is a relatively straight-forward matter to integrate supporting functionality from other applications. In this sense Tcl makes and ideal glue language. Here, a separate program, built using the *dcmtk* libraries is run which provides DICOM server capabilities. This application is able to receive DICOM data which can then be analysed using the WAS interface before being returned to another DICOM compliant application. Transfer of image data to the main tinatool shell is achieved via file I/O using a shared area of disk.

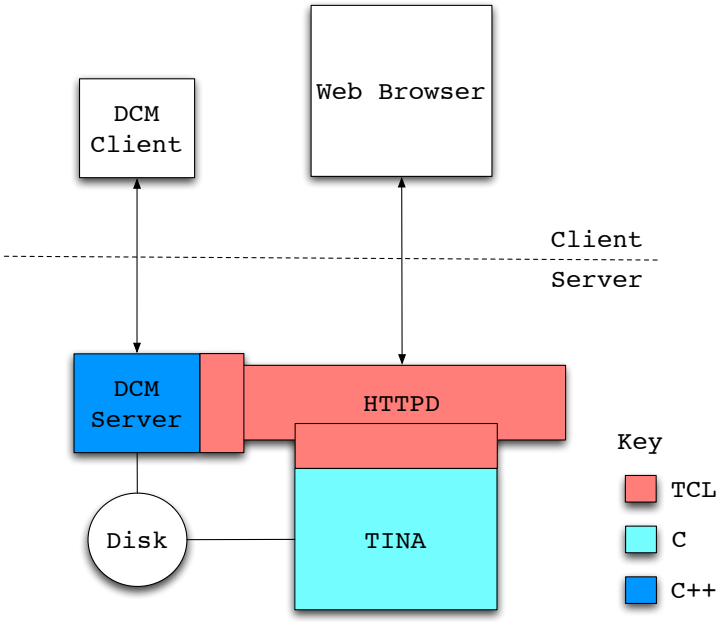


Figure 1: Overview of Web Analysis Server

3 Installation and Use

The WAS is distributed as a tina project known as the 'WebAS'. It requires that the following libraries and applications are installed and available on your system;

- Tcl libraries and include files (version 8.0 or later)
- Tclhttpd
- tina-libs (version 5.0 or later)
- tina-tools (version 5.0 or later)
- gnuplot
- libpng

3.1 Installation

1. If you have checkout the system from CVS you should first run the command;

```
./bootstrap.sh
```

2. Next you need to configure the system with the site specific information. To do this run the script;

```
./configure
```

This script will find system specific information including the locations of support librairies and header files needed to build the system. If some of your libraries are installed in non-standard locations or configure fails to find them you can provide details of these using command line options. There are many general options to configure (for a full list do ./configure –help), however some of the more useful ones include

```
--with-tina-includes=DIR          location of the tina-libs include files
--with-tina-libraries=DIR         location of the tina-libs library files
--with-tinatools-includes=DIR     location of the tina-tools include files
--with-tinatools-libraries=DIR    location of the tina-tools include files
--with-tcl                        location of the tcl tclConfig.sh script
--with-libpng-include             location of libpng png.h file
--with-libpng-library             location of the libpng library
```

For instance under MacOSX we use;

```
./configure --with-tcl=/Library/Framework/Tcl.Framwork/
```

Successful configuration completes when no **ERROR** statements are present in the summary text given at the end.

3. Once configuration is over the `tinaTool` Tcl shell can be built with

```
make
```

4. Next install the binary (the prefered location is in the bin directory within the distribution directory) with;

```
make install
```

5. You should be ready to run the system.

3.2 Running the server

First you will need to ensure that the paths set in the file `osmia.tcl` in `tclhttpd3.4.2/bin` directory point to the appropriate directories. In particular;

Variable	**Description**
osmiaDir	Full path to location of top level directory
osmiaGlbLocalDir	Full path to location of gnuplot application

If you have followed the build instructions as above then you should be able to start the server as follows;

```
cd bin
./tinaTool osmia.tcl -debug 1
```

You should see output ending with something like;

```
....
httpd started on port 8015
secure httpd started on SSL port 8016
httpd %
```

The `secure httpd started on SSL port 8016` is only present on systems where SSL has been found by configure. To connect to the server (running on a machine with the ip address 10.0.0.1) open a web browser an direct it to the URL

```
http://10.0.0.1:8015/osmia
```

This should present you with the top-level page of the analysis server.

4 System Operation

The section provides an operational overview of the web analysis server as well as providing the reader with a brief introduction to HTTP and HTML technologies underlying the WAS. For more details of these technologies the reader is directed to the many excellent texts available on these subjects.

First the user starts the server as described in the installation guide, section 3. For this example let us assume the server is running on the local machine `humphry.tina-vision.net`. A web-browser is used to connect to the server at the URL (Universal Resource Locator) "`http://humphry.tina-vision.net:8015/osmia`". This instructs the browser to send a request to the HTTP service on `humphry.tina-vision.net` which is listening on port number 8015, for the resource *osmia*.

All web browsers expect the information returned to them to be in HTML form[1]. Early web servers, served static content. That is to say, when the URL read `http://humphry.tina-vision.net/hello.html`, the web server had simply to read a file called 'hello.html' on the disk and squirt it down the HTTP channel. However, there are many tasks for which this is impractical. For instance imagine a web server which is designed to return a web page containing the current local time. Even if every possible time was stored in its own webpage, how would the browser request the appropriate page without knowing the time *a priori*? A solution to this problem (and many others) is to have dynamic content. That is, instead of the server assuming the request is a file to be read, it interprets the request as a command to execute. The server output generated by the execution of this command is used as the response to the browser. So in this case the WAS interprets the requested object 'osmia' as a command for it to execute. These operations are summarised by the diagram of figure 2.

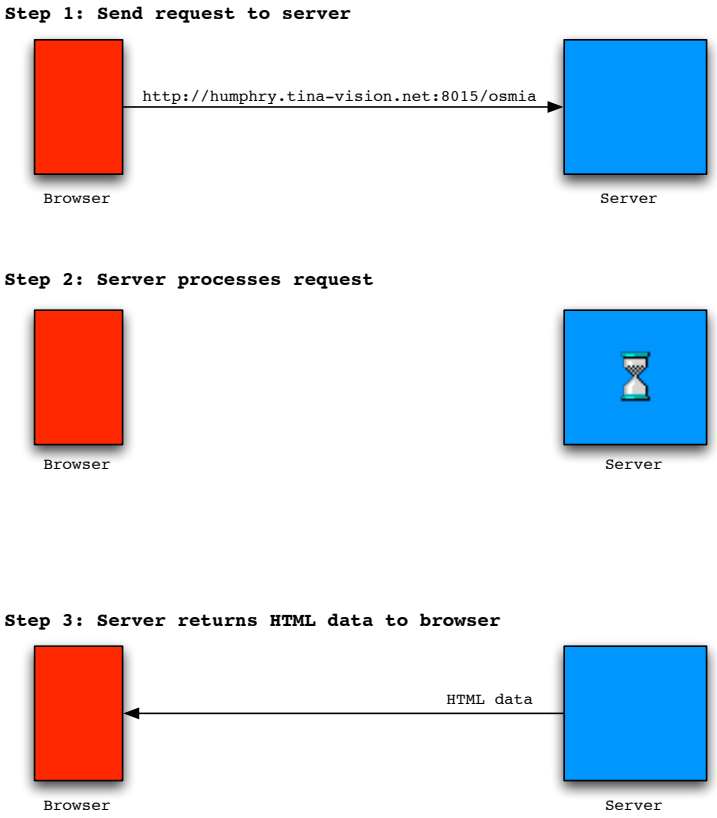


Figure 2: Summary of operations

Note the server is free to perform any operation it wishes during step 2. However, whatever it is the server does, it can only respond to the request with properly formatted HTML because this is all the browser can render (see footnoote earlier).

[1] This is a little bit of a simplification as most modern browsers actually expect a specified form of XML.

The tclhttpd server provides mechanisms for the user to extend the available functions (see section 7). The command 'osmia' is one that has been added to the server command list for its use in the WAS. When the server receives a command request it executes whatever code is specified, often including many of the non-scripted functions available in the *tinaTool* shell including those analysis functions provided by the tina libraries. This is the mechanism by which a URI request can invoke analysis operations. More properly this is known as a URI, Uniform Resource Identifier, a formatted string that serves as an identifier for a resource, typically on the Internet. Once complete the server can, if necessary, respond to the browser. This is the basis for all operations performed by the WAS.

4.1 Function arguments

The process outlined previously described a mechanism for URI requests to invoke operations in the server. This can be considered as calls to a server function, mimicking the behaviour of GUI button presses in a tradition `tinaTool` application. However, as described so far, these calls are unable to pass parameter to the functions. There are mechanisms in the HTTP protocol which can be used to pass information to the server. They are often used to support form submission. A typical web-form has many elements (entry fields, radio buttons, etc.) into which the user is required to enter data. When they have completed the form the user submits the results by hitting the 'submit' button. The information entered can then be sent to the server in one of two ways, known as GET or POST. The difference between these methods is related to the encoding, although the HTTP standard suggests that GET methods should be employed when the form processing is idempotent[2] and POST methods should be used when state information is being preserved.

The WAS uses the GET method which, technically, is the wrong method to use as repeated application of the same command can and probably will change the state of the server (see the note in 4.1.2 below). However, GET posts are very easy and convenient to use as will now be explained.

4.1.1 The GET method

When a form is submitted to the server using the GET method the variables associated with all of the form elements are concatenated onto the end of the URI using a special syntax, to be passed to the server. For instance, supposing that the a form entry set the variable named 'day' to the value 'Wednesday'. Sending this to our server `humphry.tina-vision.net` would be done as;

```
http://humphry.tina-vision.net:8015/osmia?day=Wednesday
```

Multiple variables can be included in the same URI as;

```
http://humphry.tina-vision.net:8015/osmia?day=Wednesday&month=March&year=2003
```

The server processes the incoming request to recover the identifier and variables. In the case of the WAS the identifier is used to specify the command and the variables as arguments to this command.

4.1.2 Web state-machine issues

At this point it is worth discussing the state-machine nature of web pages and web servers. The web server is typically a persistent application (running continually). As with any such application it is able to build up a set of internal states, in the simplest form values stored in variables. Thus, how the server responds to communications can be affected by the timing and ordering of those communications. Don't forget, the server could be handling requests from numerous browsers in a fashion that appears simultaneous to all users. In comparison the web browser is regarded as a transient, potentially sporadic application whose requests must be self contained, providing all information necessary to invoke the desired response from the server. This is more easily achieved by including all information in a GET type request (again this is a bit of a simplification as the POST method in combination with cookies can be used to store browser specific state information).

Generally, when state information is required when using GET mechanisms, it is passed to the server in the form of one of the variables. Specifically what is passed is the state the browser assumes or requires the server to be in, in order to correctly process the command. The server must then be able to jump to this state, from whatever condition it is in at the moment.

[2] In a pseudo-mathematical terms meaning, repeated submissions of the same data result in the same outcome.

In the case of the WAS, the server retains state information. For instance, the state of the image on top-of-stack is dependent upon the previous operation. The browser, on the other-hand, does not have a complete picture of the state of the server (for instances it does assume that the image calculator is currently being executed). It blindly issues command instructions to the server using the form interface. Thus, two users accessing the same server at the same time would certainly interfere with each other. This is simply a result of the particular implementation approach taken with the WAS. The system was designed to be a client-server application but using technologies which don't, naturally, permit/expect such coupling. Much could be done to fix this but would involve a lot of redevelopment. In the short term the easiest way to solve these issues is simply to limit the system operation to one server for each browser connection.

5 Developing and Extending the Server

5.1 Source code structure

The code for the WAS is provided as a tina project. When either downloaded as a distribution or 'checked out' of the CVS repository a directory is created called `WebAS`. At the top-level of this directory are numerous files associated with the `autoconf` build system. The important files here are ;

- `Makefile.am` : This is an `automake` configuration file. Rather than generating a Makefile directly the developer edits and updates the Makefile.am. This is then converted into a Makefile using the automake and configure utilities. The benefit is that the Makefile.am is substantially smaller and easier to understand than the resulting Makefile, which will also have many useful targets automatically defined. This mechanism also allows elements of the Makefile to be built conditioned on the results of the site specific configuration process.
- `configure.ac` : This is the configuration file for the `autoconf` system. It is written in a language known as `m4` which is a macro language. Basically it allows the developer to define a series of dependencies that the accompanying software has. From this file, autoconf will automatically build a shell script called `configure` described below.
- `configure` : This script is automatically generated by autoconf from the `configur.ac` file described above. It is an executable shell script written in `sh` the ubiquitous shell language for Unix platforms. Execution of this script will gather site specific information and use it to create Makefiles and a global include file called `config.h` which is included by all source files. The file `INSTALL` gives details of the many options that `configure` is able to except.
- `bootstrap.sh` : A script to run the autoconf tools in the correct order, thus readying the system for configuration.

The above is just intended as an overview of the magical world of autoconf. For a complete insight the reader is directed towards some of the excellent texts referenced at the nd of this document.

Many other files and directories are also visible at this level of the WAS, most of which simply support the autoconf system. However, there are two important directories. First there is the code which builds the `tinaTool` Tcl shell executable on which the server scripts will be run. This is comprised of 'C' code which links to the standard tina libraries of *tina-libs* and *tina-tools* and is located in the `src` directory. The second is the `httpd` directory which is home to the scripts which support the use of the tclhttpd system as the WAS.

6 tinaTool

At the core of the WAS is a single executable called `tinaTool`. Historically the tina libraries have always been executed using an application known as a *tinatool*. In version 5 the libraries where split into two groups *tina-libs* and *tina-tools* allowing the algorithmic code to be accessed without the graphical infrastructure of the tool environment. However, the web analysis server still uses the original tinatool application approach, with a slight twist. In normal use the tinatool application is built using one of the graphical user interface libraries in tina, such as the XView, Motif or GTK libraries, together with a supporting X11 or GDK graphics library. However, for the web analysis server the tool is built using the Tcl interface library and the NULL graphics library (for more details of the workings of the graphical interfaces to tina see the **TINA Programmers Guide**).

These provide a tinatool application in the form of a Tcl command shell with no graphical output, allowing the application to be run from the command line with no graphical dependencies and to execute Tcl structured commands. This application forms the core component of the WAS executing tina analysis functions and as the engine to the web server scripts.

6.1 Building and modifying tinaTool

The autoconf system is installed soley for the purposes of building the tinaTool application. To use the WAS project, both tina-libs and tina-tools will need to be installed. To develop with under tina (using the autoconf build system) it is necessary to have the autoconf tools installed (refer to the `README` file of tina-libs for complete details). In section 3 the process of building an executable tinaTool was described. This assumed that the source code had not been modified prior to the build and therefore the user was able to run the supplied `configure` script in order to configure the site specific details before simply making the tool. If the user wants to add or delete source files (those in the `src` directory) to the build then the `Makefile.am`'s must be modified and the configure script must be rebuilt[3].

To add or delete a file to the tinaTool build the user should follow these steps;

1. Add the file to the `src` directory. The file should conform to the coding standards outlined in the coding policy document for tina.
2. Edit the Makefile.am in the `src` directory in order to add the filename to the list of source files.. For instance suppose we wished to add the file `testTest_test.c` to the Makefile.am by extending the list of entries for `tinaTool_SOURCES`, thus;

```
tinaTool_SOURCES = tinaTool.c drawPlot_gp_graph.c drawPlot_graph.c\
                           drawPlot_hfit.c drawPlot_plot.c drawPlot_procs.c \
                           sysGen_error.c tlbaseGnuplot_tool.c tlbaseImc_tool.c \
                           wdgtsTcl_tw_choice.c wdgtsTcl_tw_sglobal.c \
                           filePng_file.c tlbaseMono_file.c tlbaseMono_tool.c\
                           testTest_test.c
                           ^^^^^^^^^^^^^^^
```

3. Move back up into the main WebAS directory with `cd ..` and run the `bootstrap.sh` script. This will rebuild the configuration files by running `autoconf` and `automake` appropriately.
4. Now configure and build the system by running the `configure` script and `make` as previously.

Similarly to remove files from the `src` area or exclude them from the build. Details of how to extend the functionality of the tinaTool by calling libraries and other resources which require modification to the autoconf system are beyond the scope of this documentation. The reader is referred to the references at the end of this document.

6.2 Tcl tina command structure

The WAS tinatool application is a command line driven application with all of the functionality of a standard Tcl shell (tclsh) but with the addition of tina specific commands. These commands are added to the Tcl interpreter in the usual way, by use of Tcl library functions such as `Tcl_CreateCommand` (the reader is directed to "Practical Programming in Tcl and Tk" by Brent Welsh for details of how to add commands to the Tcl interpreter). However, not all of the commands are added up-front before the Tcl interpreter begins accepting commands. This is because the Tcl interface in tina mirrors the behaviour of the graphic widget interfaces such as XView and Motif. More specifically the Tcl widget library in tina is a direct, drop in replacement for the XView and Motif based widget libraries. In deed with only one minor modification to the top-level *tinatool.c* file, any working tinatool can be made into a command-line Tcl shell application.

As an example consider the following `main` code extract taken from a *tinatool.c* file;

[3] In most cases modifying an existing source file will not require the `configure` script to be rebuilt

```
1  int main(int argc, char **argv)
2  {
3      tw_init(&argc, argv);
4
5      tw_tool("tinatool", 100, 100);
6
7      tw_label("Display ");
8      tw_button("New Tvtool", tool_proc, NULL);
9      tw_label("          ");
10     tw_newrow();
11
12     tw_newrow();
13     tw_label("File I/O ");
14     tw_button("Mono", mono_tool_proc, NULL);
15
16     tw_newrow();
17     tw_textsw(10, 65);
18
19     tw_end_tool();
20     tw_main_loop();
21}
```

Each call to a `tw` function is a call to one of the tina widget library functions. The tina windows widget libraries share a common interface (the `tw` commands) but have different implementations (XView, Motif, etc.). If we concern ourselves for the moment with the call to `tw_button`. In the graphical case this would result in the rendering of a button, in the current graphics panel. In the case of the call on line 8 (above), the text **New TvTool** appears on the button and a call to the function `tool_proc` with no parameters is made when the button is pressed.

The Tcl implementation of `tw_button` is quite different from the graphical versions of XView and Motif. Instead of rendering a button the Tcl code creates a new command within the current interpreter. The calling function (in this case `tw_button`), the button name (the text that would be rendered on graphical version) and the name of the tool from which the call is made are combined to create the command name. Calling this function in the interpreter will generate a call to the function specified.

Not all of the `tw` calls result in new commands. The Tcl widget layer provides implementations of all the functionality made available by the `tw` calls in tina. These calls where originally developed in the context of graphical output and therefore a degree of interpretation has been employed to map the implementations onto the functionality available within the Tcl language. In some cases the Tcl widget library provides 'lightweight' versions of `tw` functions which retain infrastructural code but lack mechanisms to rendering an aspect of the functionality outside the tina context. For example the command `tw_dialog` opens a new graphics pane in the traditional graphics libraries. However, the only important Tcl aspect of this is the name of the dialog (used to generate the command name) and the code therefore reflects this.

A summary of the important widget mappings is provided in table 1.

6.3 Command and variable naming convention

As mentioned earlier, in the Tcl implementation many of the `tw` functions result in the creation of a new command or variable in the current Tcl interpreter. A particular naming convention is employed to ensure that command or variable names are sufficiently unique (within the limit of commands created by tina), obvious and consistent with the graphical implementation. They do have a tendency to be rather long, however, the intention was to expose the functionality at the Tcl layer which would then wrap multiple calls into a script generate function.

The algorithm for naming first identifies the *toolname* in which the command or variable is to be created (i.e. from which tool the `tw` call is being made). This is then concatenated with the *name* of the button or entry field as defined in the `tw` call, separated with a '.'. The characters '_', ':', '(' and ')' are removed as they clash with Tcl syntax and the *command type* is prefixed separated with a ':'. The whole lot is then converted to lower case. In summary, a command or variable name is

tw **Command**	**Graphical Version**	**Tcl Version**
tw_button	Generate a named button on the current pane which invokes a specified function upon selection	Generate a new command which maps to specified function
tw_check	Generate a set of n check buttons from which none, some or all may be selected and update the binary value of a variable to reflect the state of the selections	Generate a new associative array of length n indexed via the check names whose values are the Boolean states ON and OFF
tw_choice	Generate a set of n radio buttons from which only one may be selected and update the binary value of a variable to reflect the state of the selection	Generate a new associative array of length n indexed via the choice names where only one indexed value is ON and all others are OFF
tw_fglobal	Generate an entry field to handle floating point values stored in a given memory location	Create a new Tcl variable whose value is to remain consistent with that of the given memory location
tw_fvalue	Generate an entry field to handle floating point values stored and retrieved via specified function calls	Create a new Tcl variable whose value is to remain consistent with that set and returned by the specified functions
tw_iglobal	As tw_fglobal but for integer values	As tw_fglobal but for integer values
tw_ivalue	As tw_fvalue but for integer values	As tw_fvalue but for integer values
tw_sglobal	As tw_fglobal but for character strings	As tw_fglobal but for character strings
tw_svalue	As tw_fvalue but for character strings	As tw_fvalue but for character strings

Table 1: Mappings between graphical and Tcl command line equivalents

constructed thus;

<**command type**>:<**toolname**>.<**name**>

For instance if the call `tw_button("New Tvtool", tool_proc, NULL);` was made from the toplevel tinatool the resulting command would be `tw_button:tinatool.newtvtool`.

6.4 Extending the command set

It is a simple matter to add new functionality to the WAS. New source files can be added to the build structure as described earlier. When developing new algorithms it is recommended that a graphical tinaTool is used. Once this is working, introduce this functionality to the Tcl version. The remaining work is to develop the scripts to support this functionality from the tclhttpd.

7 TclHttpd

Tclhttpd is a pure-Tcl implementation of an HTTP protocol server. It runs as a script on top of a vanilla Tcl interpreter or in this case the WAS tinaTool. The Tcl I/O system provides event-driven I/O facilities and a primitive that copies data from one I/O channel to another. The server does the HTTP protocol handling and then simply directs the I/O system to blast data from disk to a network socket. The Tcl runtime handles all the I/O in a non-blocking fashion so the server easily multiplexes multiple clients without the complexities of threading. The server has suprisingly good performance because of Tcl's sophisticated I/O system.

The HTTP protocol is perhaps the least interesting aspect of the server. Of more interest is the framework for generating dynamic page content, and the support for embedding the server directly into legacy applications to "web-enable" them. From a technical standpoint, a Tcl-based web server is ideal for embedding because Tcl was designed to support embedding into other applications. The interpreted nature of Tcl allows dynamic reconfiguration of the server. Once the core interface between the web server and the hosting application is defined, it is possible to manage the web server, upload Safe-Tcl control scripts, download logging information, and otherwise debug the Tcl part of the application without restarting the hosting application.

The tclhttpd code forms the framework within which the WAS has been built. Using the WAS assumes you have an available, correct installation of tclhttpd already on your system. The code included in the WAS project is only the additions to this

system required to run the WAS. The descriptions in this text relate, as much as possible, to the extra code provided as part of the WAS and do not discuss the details of the tclhttpd system itself (unless it is critical to the issue). For details of the operation of tclhttpd, the reader is referred to the website or one of the excellent texts both of which are referenced at the end of this document.

7.1 Code structure

The code for the tclhttpd side of the WAS is located in two directories in the tina WAS project area, `bin` and `httpd`

7.1.1 The `bin` directory

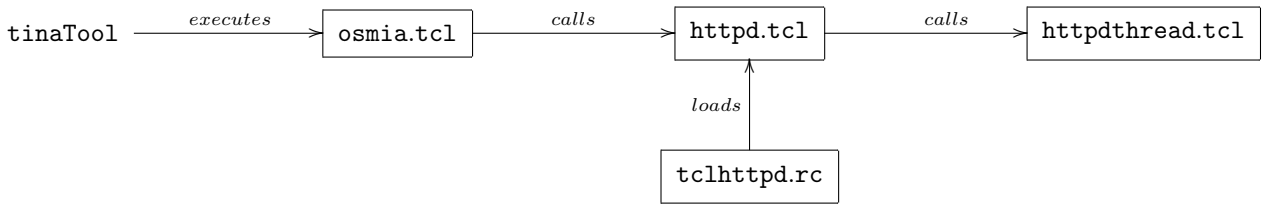
The `bin` directory contains the code the top-level scripts for the WAS. Once the tinaTool has been built it is installed into this directory. As described in the installation section 3 the WAS is run from this directory using the syntax;

```
cd bin
./tinaTool osmia.tcl -debug 1
```

This executes the script `osmia.tcl` using the tinaTool Tcl shell. The option `-debug 1` tells the server to output debug information. The `osmia.tcl` script is the top-level script and has two main purposes. Firstly it sets up a series of variables used by the rest of the WAS scripts and then it initialises the tinaTool. The second step is important to ensure that all the functionality necessary to execute the subsequent WAS scripts is available. Remember that commands are available within tinaTool once the 'tool' within which they reside has been opened (as with the graphical interface, a button is unavailable until it is visible). Finally the `osmia.tcl` script calls the main `httpd.tcl` script. The scripts `httpd.tcl` and `httpdthread.tcl` together with the resource file `tclhttpd.rc` are taken from the tclhttpd distribution. Their behaviour is of less interest to the WAS developer but is summarised below.

`httpd.tcl`	This file, which is the main startup script. It does command line processing, sets up the auto_path, and loads `tclhttpd.rc` and sourcing `httpdthread.tcl`. This file also opens the server listening sockets and does setuid, if possible.
`tclhttpd.rc`	This has configuration settings like port and host. It is sourced one time by the server during start up before command line arguments are processed.
`httpdthread.tcl`	This has the bulk of the initialization code. It is split out into its own file because it is loaded by each thread: the main thread and any worker threads created by the "-threads N" command line argument.

The call structure of the system can be summarised thus;



7.1.2 The `httpd` directory

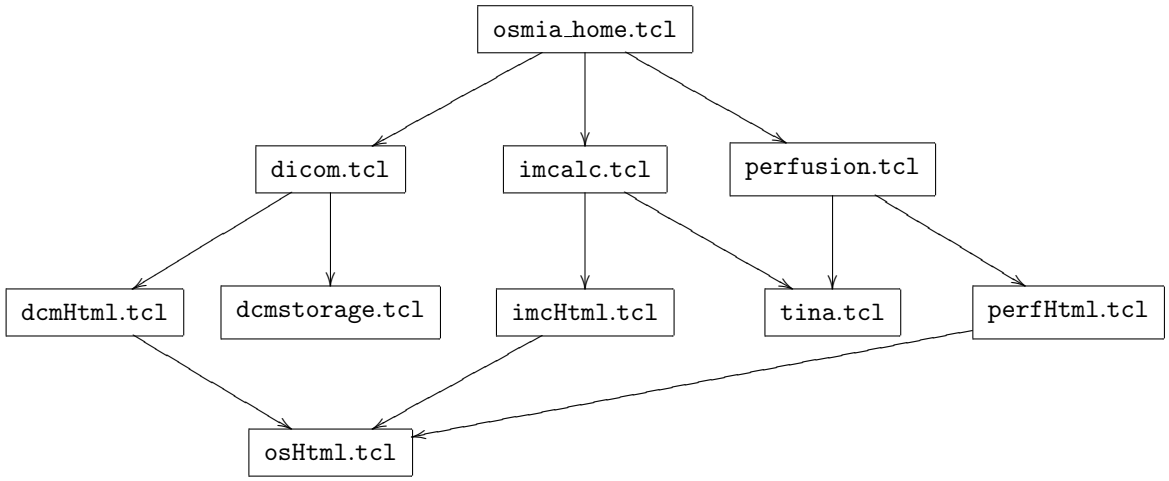
The `httpd` directory has 3 subdirectories. Of these the most important from the developers point of view is the `custom` directory. It is in here that the scripts which support the WAS are contained. Any new functionality should be added in scripts in this directory. Figure 3 is a listing of the contents as of time of writing;

Each of the top-level files of `dicom.tcl`, `imcalc.tcl`, `osmia_home.tcl` and `perfusion.tcl` provide new commands to support an area of functionality. The files in the `lib_html` directory provide support for HTML output creation, one for each of the higher level files. `lib_html` is home to more low-level interface code for the dicom interface and a command level interface to tinaTool itself, the files `dcmstorage.tcl` and `tina.tcl` respectively. Finally the files `pkgIndex.tcl` are Tcl resource files which are used simply to manage the structure.

```
+--custom
       +--dicom.tcl
       +--imcalc.tcl
       +--osmia_home.tcl
       +--perfusion.tcl
       +--lib_html
              +--dcmHtml.tcl
              +--imcHtml.tcl
              +--oshtml.tcl
              +--perfHtml.tcl
              +--pkgIndex.tcl
       +--lib_model
              +--dcmstorage.tcl
              +--pkgIndex.tcl
              +--tina.tcl
```

Figure 3: `custom` directory

`osmia_home.tcl` provides the functionality for the WAS start page. From here the dicom server, image calculator and perfusion analysis systems can be accessed. This dependency relationship between these files is summarised below;



7.1.3 Other directories

Temporary HTML document components are stored in the `htdocs` directory. This directory is distributed with a single file, used in display when no images are available in the image calculator. The directory listing of this is given in figure 4.

```
+--htdocs
       +--empty.jpg
```

Figure 4: `htdocs` directory

The `html` directory is used to store permanent HTML document components. In this area one directory in particular is of note, `text`. In here are a set of Tcl scripts which do little more than assign text to variables that can be used during the HTML generation. In this case it is the perfusion text.

```
+--html
      +--dicom
            +--PERF
            +--SAMPLES
                  +--CT-MONO2-16-ankle
                  +--CT-MONO2-16-brain
                  +--CT-MONO2-8-abdo
                  +--MR-MONO2-12-an2
                  +--MR-MONO2-12-angio-an1
                  +--MR-MONO2-12-shoulder
                  +--MR-MONO2-16-head
                  +--MR-MONO2-16-knee
                  +--MR-MONO2-8-16x-heart
                  +--OT-MONO2-8-a7
                  +--OT-MONO2-8-colon
      +--plot
      +--text
            +--homeTxt.tcl
            +--perfStep1Txt.tcl
            +--perfStep2Txt.tcl
            +--perfStep4Txt.tcl
            +--perfStep5Txt.tcl
      +--tina
            +--perf
                  +--perfBinMap.pgm
                  +--perfBinThresh.pgm
                  +--perfCBV.pgm
                  +--perfCBVHist.png
                  +--perfCBVMap.pgm
                  +--perfCBVMapHist.png
                  +--perfFirst.pgm
                  +--perfFirst.png
                  +--perfFirstHist.png
                  +--perfGammaFit.png
                  +--perfHist.png
                  +--perfPlot.txt
                  +--perfTTM.pgm
                  +--perfTTMHist.png
                  +--perfTTMMap.pgm
                  +--perfTTMMapHist.png
                  +--plot.inf
            +--stack
                  +--dummy.png
```

7.2 Mechanisms

In order to describe the mechanisms which enable the WAS to receive and process the commands from the browser we shall dissect some of the Tcl files. For more detailed analysis of these processes the reader is referred to the tclhttpd documentation provided with the software as well as references at the end of this document.

7.2.1 osmia_home.tcl

```
if {[catch {package require oshtml}]} {
```

```
    return
}
```

The above code segment loads the HTML output support functions (from the file `oshtml.tcl` file in the `lib_html` directory). The `package` command is an inbuilt Tcl function which locates and loads the package using information in the `pkgIndex.tcl` files.

```
Direct_Url /osmia    ::osmia::
```

This line maps the URI to a Tcl procedure. In other words, this line is the one which makes it all possible. In this case when the URI `/osmia` is received the procedure `::osmia::` is called. In fact *any* URI which begins `/osmia/` will be handled by `::osmia::`. In fact `::osmia::` is a Tcl *namespace* which is a mechanism used to segment procedure naming, allowing two procedures with the same name to exist within the same program. In this case the procedure is actually `/`. So when the URI ends with `/osmia/` the procedure `/` in the namespace`::osmia::` will be called.

```
namespace eval ::osmia {
proc ::osmia::/ {args} {
    set html    [oshtml::header "OSMIA Server - Home Page"]
    append html [oshtml::home_table]
    append html [oshtml::links_list home]
    append html [oshtml::footer logo]

    return $html
}
```

The above declares the procedure `/` as a member of the namespace `::osmia::`. In this case although the arguments to the function would be available to the programmer as the Tcl list `args`. This procedure uses calls to the `oshtml` package (which shares the same namespace name) to create a the HTML output. This output is created within a Tcl variable, `html`. Each of the `oshtml` procedures returns part of the page from the header to the footer. Finally this is returned which sends the output back to the browser.

```
proc ::osmia:: {args} {
    ::osmia::/
}
```

This function simply ensures that if the trailing `/` is missing from the `/osmia/` the correct function is still called.

7.2.2 imcalc.tcl

This file is the main one responsible for generating the HTML form interface for the image calculator. A good example is the `::imcalc::/func` procedure, presented below.

```
proc ::imcalc::/func {args} {
    set op      [lindex $args 1]
    set param_1 [lindex $args 3]
    set param_2 [lindex $args 5]
    set param_3 [lindex $args 7]
    set param_4 [lindex $args 9]

    if {[string compare $op "Undo Last"] != 0} {
        tina::stackSwapDup
    }
```

This part of the procedure extracts the name of the calculator function required by the user as well as any arguments to that function. This information is provided to the system via a GET HTTP message (see earlier). Such requests were generated from the image calculator HTML page, created by `imcHtml.tcl`.

```
    switch -exact -- $op {
        "Undo Last" { tina::funcUndo }
        invert  { tina::funcInvert }
        coil    { tina::coilCorrect }
        square  { tina::funcSqr }
        sqrt    { tina::funcSqrt }
        log     { tina::funcLog }
        exp     { tina::funcExp }
        sin     { tina::funcSin }
        invsin  { tina::funcAsin }
        diffx   { tina::funcDiffx }
        diffy   { tina::funcDiffy }
        grdsqr  { tina::funcGrdsq }
        lap     { tina::funcLap }
        ddn     { tina::funcDdn }
        ddt     { tina::funcDdt }
        median  { tina::funcMed }
        tsmooth { tina::funcTsmooth }
        noise   { tina::funcNoise
                  setNoiseLevel [expr 2 * [tina::getConst]]
                  setFactor     [expr 1 / [tina::getConst]]
                }
        add_k   { tina::funcAddConst $param_1 }
        mult_k  { tina::funcMultConst $param_1 }
        thresh  { tina::funcThres $param_1 }
        sample  { tina::funcSample $param_1 }
        gauss   { tina::funcGauss $param_1 $param_2 }
        rank    { tina::funcRank $param_1 $param_2 }
        linseq  {# Set parameters using radio button args

        if {[string compare $param_2 up] == 0} {
                    set param_2 0
            } else {
                    set param_2 1
            }
                  if {[string compare $param_3 left] == 0} {
                      set param_3 0
            } else {
                      set param_3 1
             }

         tina::funcLsf $param_1 $param_2 $param_3
        }
    }

    ::imcalc::/ imcOp doOperation
}
```

The remainder of this function is used to invoke the appropriate procedure (in the `tina.tcl` file) using a switch statement. Finally it regenerates the image calculator HTML page.

7.2.3 `imcHtml.tcl`

Calls to `imcHtml.tcl` are used to generate the HTML output which tells the browser how to render the image calculator page as shown in figure 5. Much of this page is rendered using HTML forms as the examples next show.

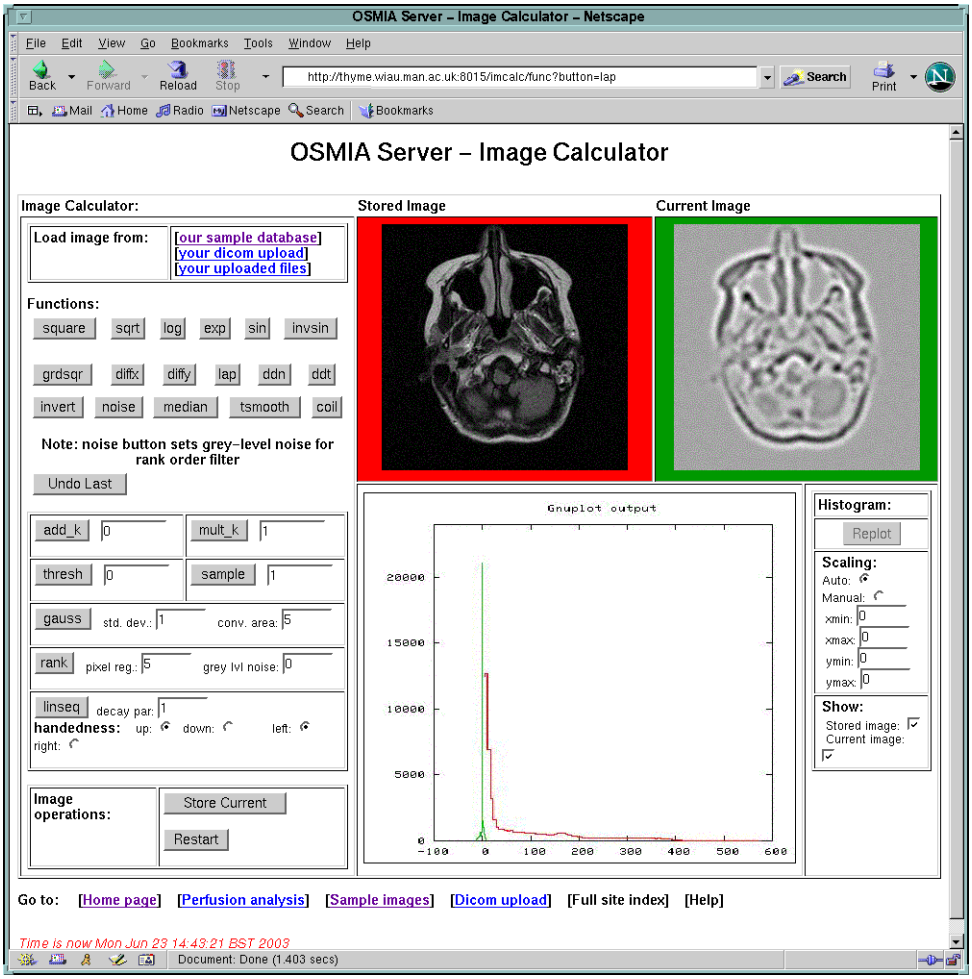


Figure 5: Image Calculator Web Page

```
    ...
1    append html "        <form action=\"/imcalc/func\">\n"
2    append html "        <br><big><strong>Functions:</strong></big><br>\n"
3    append html "        <table border=\"0\" width=\"100%\" cellpadding=\"3\">\n"
4    append html "          <tbody>\n"
5    append html "            <tr>\n"
6    append html "              <td>\n"
7    append html "                [oshtml::inputSubmit button square]  \n"
8    append html "              </td>\n"
    ...
```

Above is a segment of the procedure `imcHtml::displayImcalcForm`. It is this function which is responsible for constructing the main button interface of the image calculator (as seen to the top left of the figure 5). The above section is used to create the “square” button. Line 1 sets up the form action, which is the URI to send the form submission details to (and thus the mapped Tcl function to invoke). Lines 2-6 generate the table and table cell. Line 7 is responsible for generating the button. Actually, it calls the `inputSubmit` procedure with the request for a button called square, see next.

7.2.4 osHtml.tcl

The inputSubmit procedure is given below. This simply generates the necessary HTML to create a button instance.

```
proc oshtml::inputSubmit {name value {title Bob} {enabled true}} {
    set    html "<strong><input type=\"submit\" name=\""
    append html $name
    append html "\" value=\""
    append html ${value}"\"
    append html "\" title=\""
    append html ${title}"\"
    if {[string compare $enabled true] != 0} {
        append html " disabled"
    }
    append html "></strong>"
    return $html
}
```

And this completes the picture. The button instance is created here and rendered by the browser. The user selecting the button will communicate with the server using a GET HTTP message. The content of this message is generated by the details in the button. These details specify the Tcl command to call (in this case a procedure). This procedure will ultimately invoke a tina function available within the tinaTool shell.

8 Suggested Reading

The following is a list of suggested resources which the reader might find useful.

8.1 Useful web sites

1. http://www.tina-vision.net :The main tina web pages.
2. http://developer.tina-vision.net :The tina developer web pages (useful for wiki and mailing list archives).
3. http://www.cvshome.org/docs/ : Useful documentation for using CVS including the original ”Cederqvist” manual which can be read online or downloaded.
4. http://developer.tina-vision.net/cgi-bin/wiki.pl?UsingCVSWithTina : The tina CVS guide on a wiki page.
5. http://www.gnu.org/manual/autoconf/index.html : Autoconf manual for automake, autoconf etc all freely available
6. http://www.tcl.tk/ : The home Tcl/Tk the scripting language used for the WAS. Freely available for download.
7. http://www.tcl.tk/software/tclhttpd/ : The Tcl based HTTP server used as the basis for the WAS. Freely available.
8. http://wiki.tcl.tk : A wiki site for Tcl.
9. http://dicom.offis.de/dcmtk.php.en : DCMTK. The dicom toolkit used for the dicom server.

8.2 Books

1. **HTML The Definitive Guide**, *Chuck Musciano & Bill Kennedy*, 3rd Edition, O'Reilly ISBN 1-56592-492-4.

2. **The C Programming Language**, *Brian W. Kernighan & Denis M. Ritchie*, 2nd Edition, Prentice Hall, ISBN 0-13-110362-8.

3. **Practical Programming in Tcl and Tk**, *Brent B. Welch, Ken Jones with Jeffery Hobbs*, 4th Edition, Prentice Hall, ISBN 0-13-038560-3. *Note this includes a chapter devoted to the tclhttpd so is well worth a read*