# The Active Notebook



## 1. Introduction

The Active Notebook is a web-based technology for a computer-based notebook which monitors and records SeisRes tasks as they are being performed by the user. Past work in SeisRes can be reactivated and investigations can be renewed using

the Notebook. With the ability to author new web content linked closely to the SeisRes computation that generates it, Notebook has the ability to author content and carry out SeisRes tasks using a document generation metaphor. The Notebook serves as the main user interface to a broad set of tasks, broad because it is infeasible to track detailed internal user activities while using interactive programs such as EarthGM. However, SeisRes tasks and the SeisRes SDV can be initiated and captured in the Notebook. Currently we have implemented two task-submission example documents, one submitting a 3D-EFDSS seismic modeling task to the Lamont SP2 using loadleveler and one submitting a set of fluid flow simulation tasks.

## 1.1. What the Active Notebook Does

- Archives user's workflow
- Provides browsing of past workflow
- Provides interface to new workflow submission
- Provides interface to monitor workflow status, progress, performance, and steering
- Provides interface to other than SeisRes computation tasks such as repository management
- Hypertext scientific notebook functionality
- Interface to versioning and version management
- Provides authoring interface of new notebook content
- Provides some view of workflow and data derivation

### 1.1.1. What the Active Notebook Does Not Do

- Track execution within apps like EarthGM
- General undo

### 1.1.2. Features of the Active Notebook

- Mixed Initiative -- the server side can take initiative of notebook contents as well as the end user. Authoring notebook content is synergistic between the user and the SeisRes system. For example, ongoing monitoring of partial results in the notebook can be driven by events using server push.
- Extensible System -- Both server and client support scripting so new content can be authored at runtime. New documents can be added at any time.

Plugins in browser will either be scripting based or have scripting-based interfaces.

- Supports Different Presentations -- The notebook will be able to present the same object in different ways -- for example a spreadsheet grid representation or a 3D visualization of a well path.
- Publish Reports -- It is a goal that the notebook will generate reports that can be given to the client. These reports can either be viewed from a web browser (with requisite plugins) or they can be printed.
- Authoring Interface -- This interface will allow the user to quickly construct experiments, to save and restore both the steps of an experiment, and to generate all of the script-based code to that is necessary to run the experiment. It also allows for new contents representing new steps or designs to be added by knowledgeable content authors.
- Status Interface -- The status display will provide a "birds-eye" view. The status display also contains synopsis (hyperlinked to details) of final results of an experiment and any given step within an experiment. The status display provides a scientific or computational presentation (progress in pipeline, status, performance, etc) of partial results of an experiment or its steps. The notebook can provide for steering of ongoing computation.
- Versioning -- the notebook can provide an interface to versioning and the ability to create new version branches of an experiment.

## 1.2. Major subsystems of this component

There are four main subsystems, the notebook server, the seisres server, notebook clients, and the application pool.

## 1.3. Languages and architectures used by this component

The notebook client uses the seisreswish tcl image and the tcl in the tcl plugin as well as dynamic modules. SeisRes functions are C++ wrapped with swig and available in tcl.
The notebook server uses APACHE (written in C/C++) and Zope (C/Python) as well as SeisRes functions wrapped with swig and available in PYTHON (not used at this time though). A tcl-based cgi script is used.
The application pool uses the languages that the applications are written in and shell scripts including tcl. If they use wrappers, then it also uses the SeisRes tcl wrap. Python and perl are also avalaible for scripting.
SeisRes server functions are described elsewhere.

## 1.4. Executive summary. Diagram illustrating relationship to other software components.

The notebook server uses a web server, APACHE and a web applications sever called ZOPE. Both are open source products. The server can use the PYTHON encapsulation of SeisRes functions since it is based on PYTHON. It is not necessary to use APACHE when running Zope as it can use its own server.
As off December 1999, recent versions of Apache and Zope -- Apache 1.3.9 and Zope 2.1 have been used.
The notebook client consists of a Netscape Web browser and the tcl plugin (either the NASA one or the one available from Scriptics.) There are functions that will not work with the tcl plugin within Internet Explorer.
The tcl plugin is version 2 and the NASA plugin is 20.1.
The client side makes use of many seisres functions using TCL scripts -- often using the **seisreswish** executable. In addition the work scripts it generates for computers in the work pool also makes use of seisres functions often using **seisrestclsh.** The plugin is based on tcl 8.0.2. We are currently using tcl 8.0.5 for the tcl scripts.

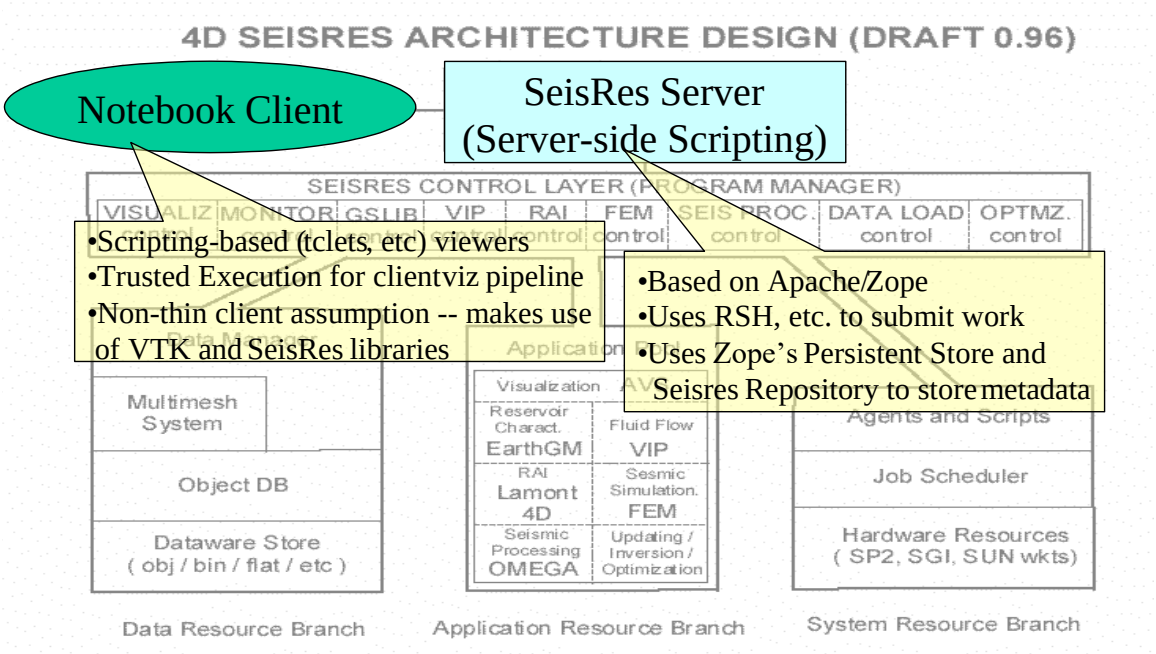


***Figure 1**. Proposed Notebook System Design*

## 1.5. List relationship to any 3rd party software for this component.

The notebook server uses APACHE (written in C/C++) and Zope (C/Python) as well as SeisRes functions wrapped with swig and available in PYTHON (not used at this time though). A tcl-based cgi script is used. It also uses the tcl plugin, the broadway plugin and the metagui.

### 1.6. Describe the typical use with an application and provide a brief code example.

See section, 2.2.6: Current Notebook Implementation Notes.

## 2. Notebook Architecture and Major Parts

### 2.1. Notebook Architecture

For the past year we have been prototyping aspects of an active notebook, some of which were demonstrated at the October 1998 SeisRes meeting and some that has been developed more recently. The proposed architecture displayed in Figure 1 is the result of these investigations.
The gist of the architecture is to have server side scripting dynamically generate user interfaces to SeisRes tasks within the notebook. (These interfaces could replace or only mirror a standard client-server GUI to SeisRes -- the design is flexible enough to accommodate both.) These interfaces are archived within the notebook as work steps of SeisRes progress. A not-so-thin client will support visualization pipelines (see visualization document) visualization being embedded in a web browser. The notebook is active in allowing reviving of past work and re-interacting with it. We make use of standards, working standards, and emerging standards to "ride the software wave" when feasible.
Design Note: We make use of both server-side scripting, and client-side scripting to make the notebook active. The spirit of design is to be able to dynamically create from SeisRes metadata in persistent store, new interfaces to SeisRes workflow tasks. Scripting on both client and server are coordinated to achieve this. We prefer scripting languages to a language that is compiled or even byte compiled (i.e. Java) for this task. (We are using tcl and the tclet plugin presently.) While a SeisRes notebook document may include Java applets these are placed (authored) on the web document using scripting. Note however that there are scripting language interpreters that work within Java. Some examples: FESI, a free EcmaScript (Javascript-like standard effort supported by Netscape) Interpreter in Java, or JPYTHON, a Python Interpreter in Java.

### 2.2. Notebook's Major Parts

#### 2.2.1. Notebook Client

The client may support other local computation orchestrated though the web browser as well. Currently, the prototyping of the notebook is based on using VTK as

a tclet plugin. The browser will have an interface to SeisRes repository objects directly within the browser's scripting environment(s). These may include, but are not limited to, tcl, Javascript, or JPython. The scripting environments in the browser have access to the browser's document object model (DOM). One such access is Netscape's LiveWire. Microsoft has their own DOM and DOM-interface to their server-side and client-side scripting. An emerging standard is W3C's DOM. There also may be a need for applets and plugins to communicate between each other. One way to do this is to use the SeisRes server as a communication relay. This has an advantage that the server can track more immediate changes of state. See the section below that describes the notebook communication needs. Also see the included proposal by Jody Winston for an XML-based way to specify GUI's based in a web browser. The Mozilla work referenced in that document includes an Application Object Model (AOM) as well. Clearly this is the way to go in the long run. This technology is not mature enough at this time, however.

The not-so-thin client will run in trusted mode when executing the visualization pipeline as well as interfacing to repository and event objects. This means that there is very little protection on what embedded-in-browser code can do. Since SeisRes will be used in an intranet situation, where others are trusted, protection of what runs on the client is the same as the level of trust as running a normal application. Normal Internet trust will be applied to normal browsing other than SeisRes.

We propose, in the long run, the use of XML to transport structured data between the browser and the SeisRes server. The browser will make use of embedded viewers that parse XML for viewing or parse the XML directly in their scripting. Embedded viewers can be script based (i.e. tclet), Java based, or pre-built as a plugin. Our current implementation does not make use of XML.

The client notebook will support an interface to authoring and status. Authoring will be a mixed initiative approach with requests to the SeisRes server (using http POST for example) from the user generating new notebook content using server-side scripting (i.e. 3-Tier design). We currently use the Zope authoring interface for this.

### 2.2.2. SeisRes Server

The server tracks the workflow progress of a user through the SeisRes steps and dynamically constructs new web content including client-side scripts based on user initiative, SeisRes objects, and metadata about the state of workflow of the on-going SeisRes experiment. Changes in state in the client are tracked with forms submission (http POST), cookies, and direct plugin communication with the server (for example, the ability for a tclet plugin to do http POST). The ultimate store of persistent data will be the SeisRes metadata store (in Zope and the SeisRes repository) -- cookies may be used, however, to stage persistent data to this store. We use Apache as the http server. We run the python-based Zope under Apache as a way of dynamically publishing objects to the web. The idea is to use Zope objects to represent the workflow tasks and documents of SeisRes. These objects represent task submission, monitoring, results, synopsis, and associated setup documents that contain submission forms for parameter files for example. The idea

is to apply a model of document preparation into a document set that represents a SeisRes experiment as the way to organize and execute the SeisRes workflow. Preparing SeisRes documents will execute SeisRes work tasks on the set of computational resources for SeisRes work. Zope has a persistent object system so these documents are archived with the state of SeisRes work though its work tasks. Zope uses a server side scripting language called DTML.
The main functionality of the server side scripting is to construct client-side scripts and web content that comprises:

- an interface to SeisRes tasks dispatched from the SeisRes Server to compute hosts using RSH and SSH (preferred).
- the generation of a client side script that can generate visualization (VTK) pipeline network or a user interface on the no-so-thin client for example.

### 2.2.3. Metadata Store

Persistent metadata will be stored both in Zope's persistent store (mostly) and the SeisRes Repository. Versioning metadata, such as what comprises of an experimental fork, will also be stored here. SeisRes objects do have version attributes as well.

| Currently the SeisRes repository can store associative arrays of variable/values and we are transitioning to storing values that are set in forms to using this mechanism. Also, the SeisRes repository can store environment variable/values so the environment variable properties will be stored there -- some of these are stored as properties in the notebook currently. Lastly, some of the variable metadata could be stored in the wrapper specs in the repository |
|---|

### 2.2.4. Notebook Communication

Figure 2 summarizes graphically the communication needs between notebook components and the SeisRes server.
To summarize:

- Applets/plugins need to see the DOM (Document Object Model)
- Applets/plugins need to see each other
- Server Side Scripting may need to see the DOM
- Applets/plugins can post information to the server
- Client-side Scripting and Applet/Plugins can access Events and Repository objects directly using the Event and Repository servers. They can communicate with Zope metadata using http POST and indirectly with server side scripting.

# Notebook Communication

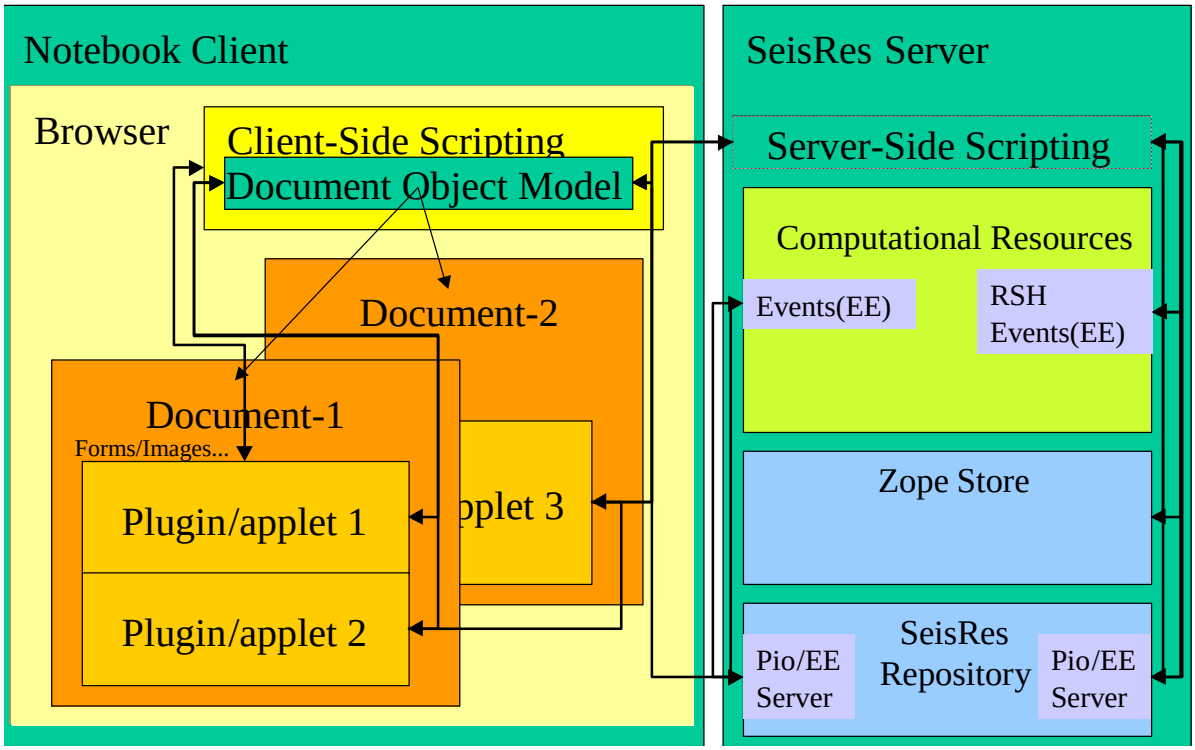


**Figure 2.** Notebook Communication Needs

### 2.2.5. Order out of Chaos -- Document Management and User Interface

Good user interface design should be heeded in how the user uses the active notebook in doing SeisRes tasks. This includes issues on organizing the sum of work for a SeisRes experiment to the layout of menus and user input widgets in plugins, applets, etc on an active notebook page. Individuals approach complex tasks in different ways and the overall SeisRes system need to accommodate user-varying initiatives on work order, etc. Publishing SeisRes results to a wider community including the ultimate customer has implications on how the notebook content is managed and disseminated.

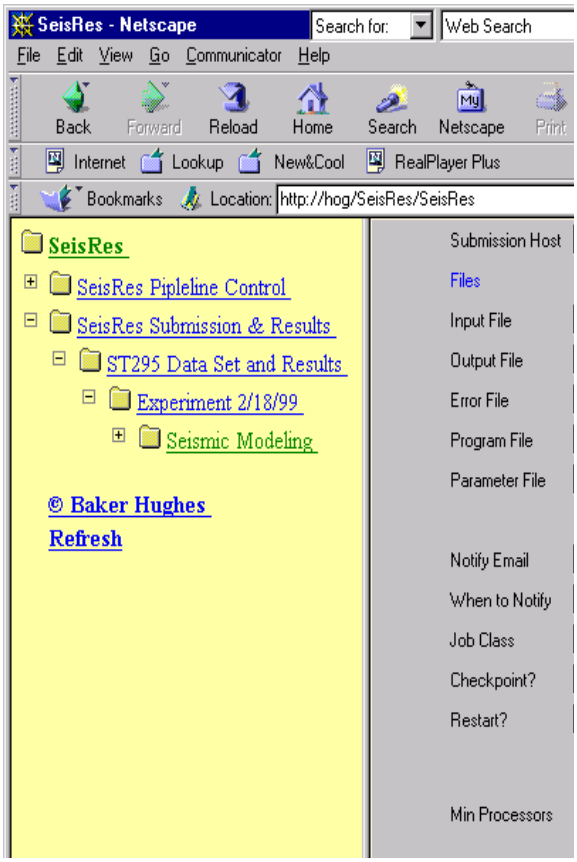


*Figure 3. Zope tree display used to organize documents*

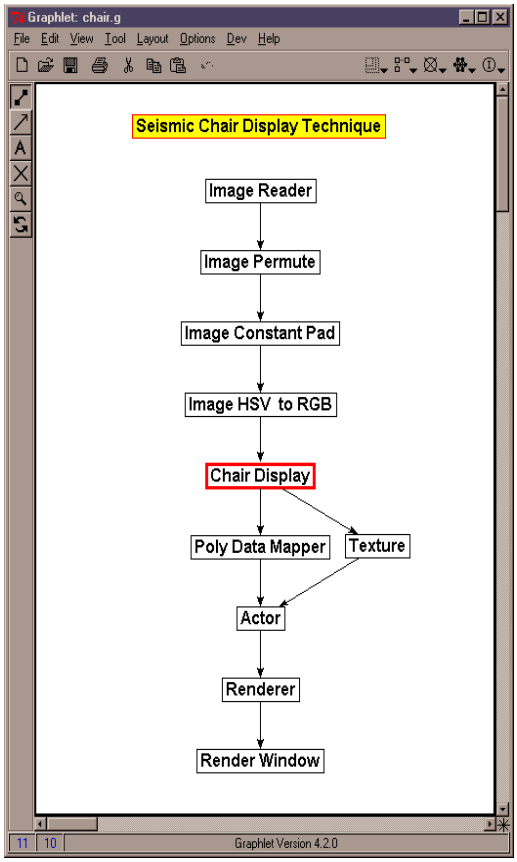


*Figure 4. Dataflow Diagram of Chair Display VTK Pipeline*

To address these concerns we implement two features:

*Introduce a notion of viewers*, which are plugins, applets etc that have careful, simple interface design but covers the major functions a user would do with the viewer. (Other less frequent options controlling more external-to-the-task context could be entered using html forms.) A viewer could be something like a seismic viewer, which could be a carefully designed intuitive interface to visualizing seismic data. The parameter widgets and menus with the viewer would have a layout that considers the human factors involved in a user visualizing seismic data. A viewer could also not do much visualization but act more as a control element. A balance needs to be struck between the flexibility of building web content on the fly from a knowledge base and a considered design for user interaction in common SeisRes tasks. We propose the notion of a viewer to address this balance. In our current implementation we use the tclet plugin as the basis for the viewers.
*Introduce a web document management and dissemination system* in the SeisRes server. Explorer  (as in Windows) like interfaces and/or dataflow graphs will be used to give the user a high level view of the notebook contents with drilldown as well as steps to be executed in a pipeline. .  In our current implementation, we use Zope as the document management system. The two figures (3,4) show the use of Zope's DTML tree to organize the notebook contents of a SeisRes investigation and the use of the tcl-based graphlet to show the dataflow of the pipeline for the seismic chair display. (The graphlet example is not yet implemented currently, but it is our intent to provide interfaces like this for workflow.)  *We now have a basic package for graphing workflow.*

### 2.2.6. Current Notebook Implementation Notes

We will walk though how a SeisRes task is implemented in the notebook. The figure below is a picture of the Index page for the Seismic Modeling task in SeisRes.  The Left part of the window contains a tree view of the documents comprising SeisRes work. Note that the work for different projects and experiments within projects go into the SeisRes Submission and Results folder. Within that folder is one Project, ST295, one Workflow, Experiment 2/19/99 and one SeisRes Task, Seismic Modeling. Of course, a complete working SeisRes will contain many more Datasets, Workflows and Tasks. The right window displays an index of documents for the Seismic Modeling Task.  These documents are generally a task submission document, an ongoing work monitoring document when appropriate, a results document, and any auxiliary documents -- in this case a Seismic Chair Display document. The experimenter is free to author more than the standard set of documents – in fact Zope provides easy ways to add document products like threaded discussions at any point in the document tree.

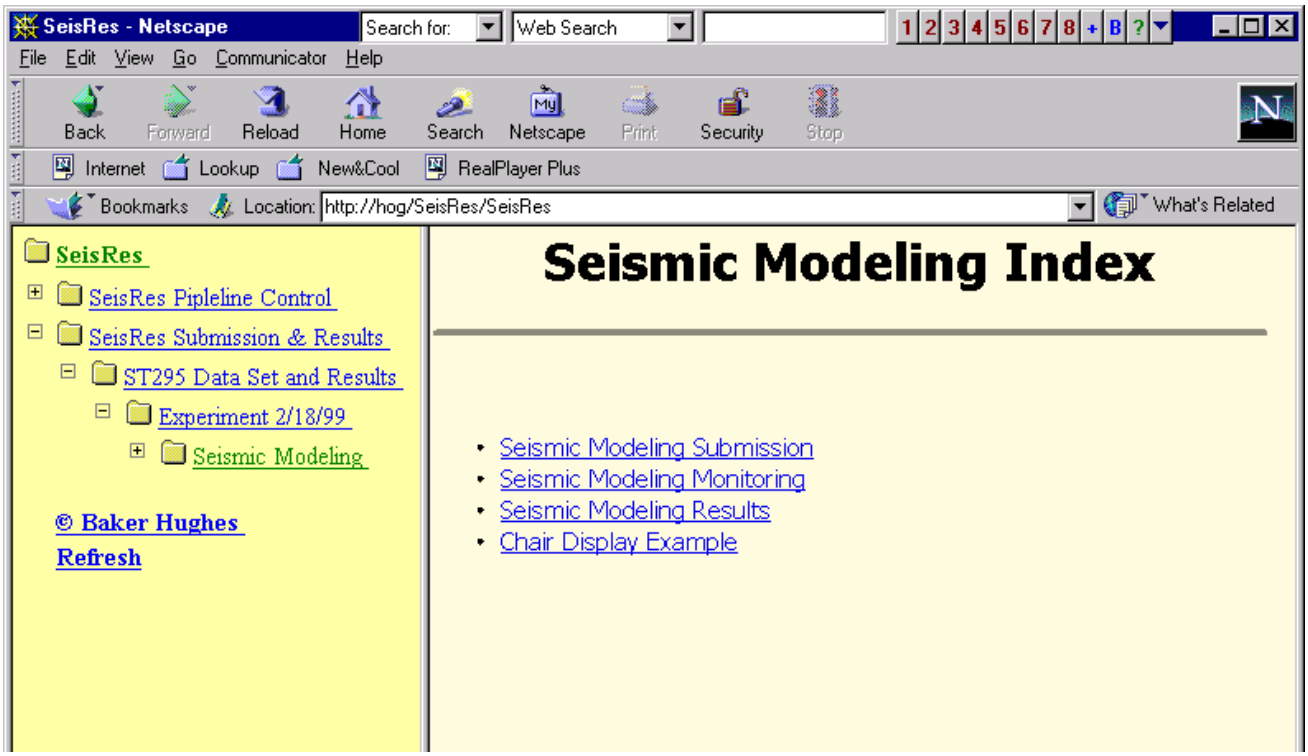


***Figure 6.*** *Active Notebook Inteface to the Seismic Modeling Task*

Currently, we have implemented two task submission example documents, one submitting a Seismic Modeling task to the SP2 using loadleveler and one of Wei's Flow Simulator tasks. Monitoring and Results examples (using events and Wendell's SDV to visualize -- see Appendix B for an example of VTK visualization in a Netscape browser.) are still to be completed. We outline below the details of how the Seismic Modeling Submission document is implemented. The next figure shows the Seismic Modeling Submission form. Looping though metadata stored in a subfolder of the submission folder for Seismic Modeling creates this form. The metadata for the variables include past values from the prior submission, type info, default values, min and max, etc. Another DTML document lays out the variables into the tcl-based tclet form display. The client-side form display uses a tcl/tk package called metagui to help do its work. The user can restore the values to default values (erasing the last value) by using the restore defaults button. When the user presses submit, an http POST goes to the Zope server executing a DTML document that processes the submission settings of the variables. It calls a simple PYTHON method to substitute place holder variables with (the %%-delimited names shown in Table 1) the values from the POST in a script template. It then copies this script to a temporary file and remote copies, then RSHs the script using loadleveler on the SP2. The submit program is a regular shell-like csh for work (such as the Eclipse task) on machine that can execute the work immediately. The submit program is specified as metadata (a Zope property) of the Submission folder. Figure 7 shows the Submission Response Page.

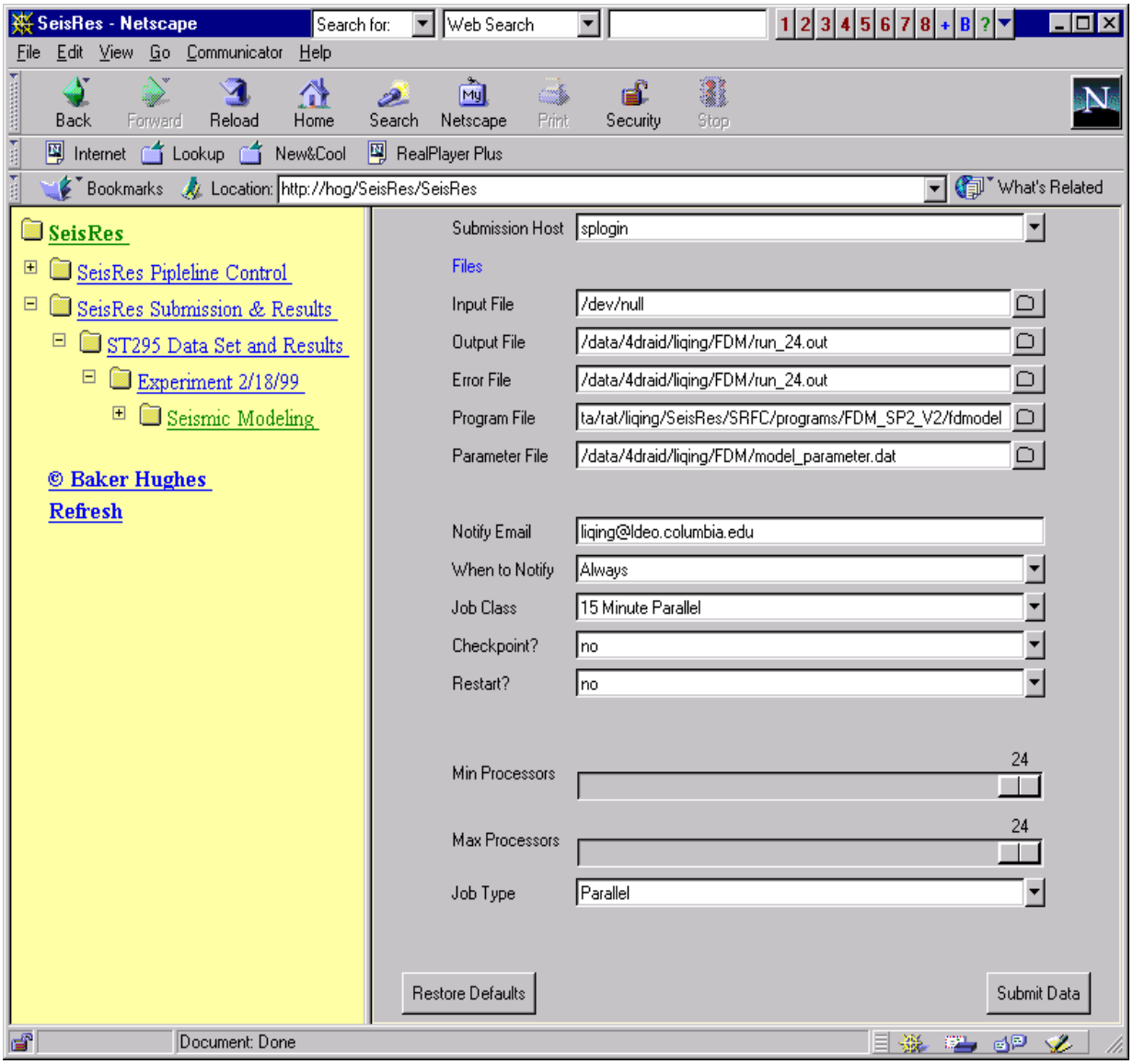


***Figure 6.*** *The Seismic Modeling Submission Form.*

### 1. The script template for the Seismic Modeling Task.

```
#!/bin/ksh
# @ input = %%input%%
# @ output = %%output%%
# @ error =  %%error%%
# @ notify_user = %%notify%%
# @ class = %%class%%
# @ notification = %%notification%%
# @ checkpoint = %%checkpoint%%
# @ restart = %%restart%%
# @ requirements = (Arch == "R6000") && (OpSys == "AIX42") && (Adapter == "hps_user")
&&  (Pool == 01)
```

```
# @ min_processors =%%min_processors%%
# @ max_processors =%%max_processors%%
# @ job_type = %%job_type%%
# @ queue

export MP_PROCS=24
export MP_RMPOOL401
export MP_EUIDEVICE=css0
export MP_EUILIB=us
export MP_STDOUTMODE=ordered
export MP_INFOLEVEL=3
export MP_LABELIO=yes
export MP_STDOUTMODE=ordered

/usr/bin/poe %%program%% %%parameter_file%%
# send notice
# mail %%notify%% < %%output%%
# mail %%notify%% < %%error%%
```

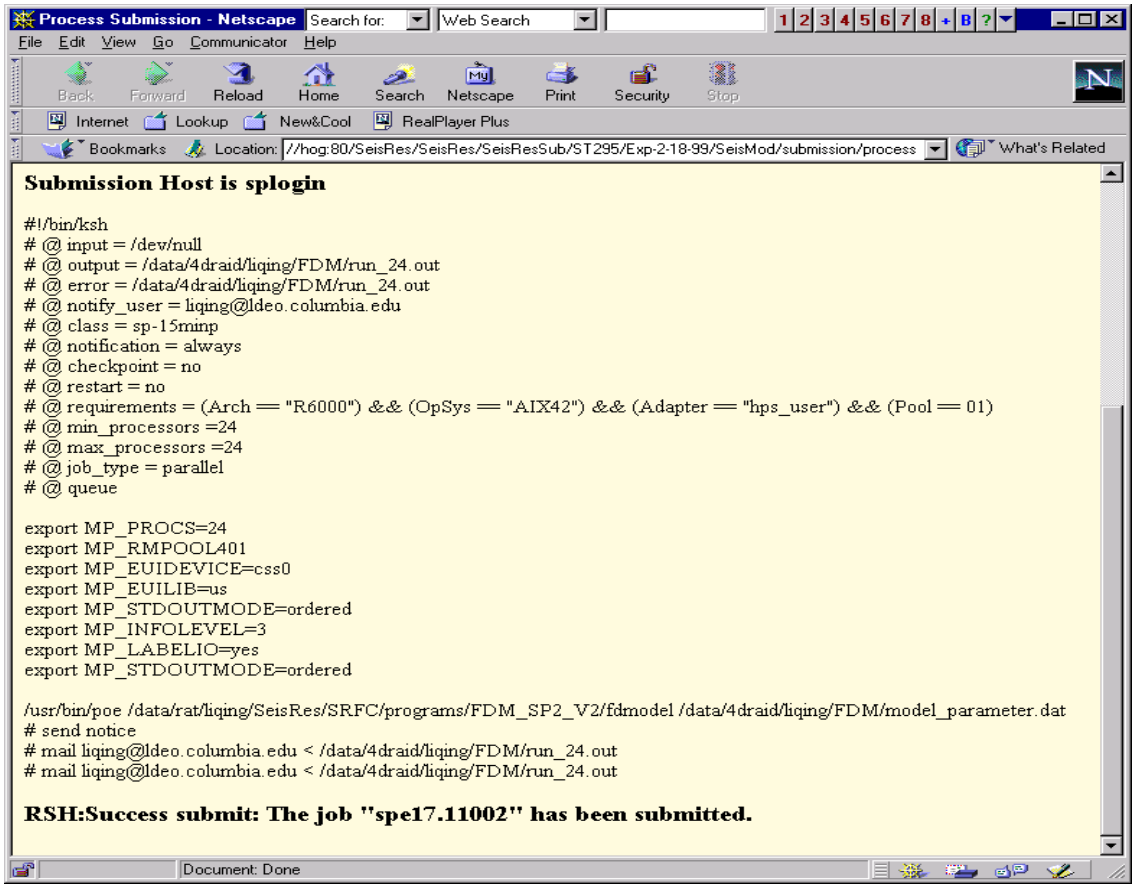


**Figure 7.** The Submission Response Page

.

Figure 8 shows the DTML script that processes the http POST data. The Zope authoring interface is shown. This DTML script uses two PYTHON methods, makeScript to do the variable substitution using the script template, and submitIt to do the remote copy of the script and its RSH execution using the summitprog.

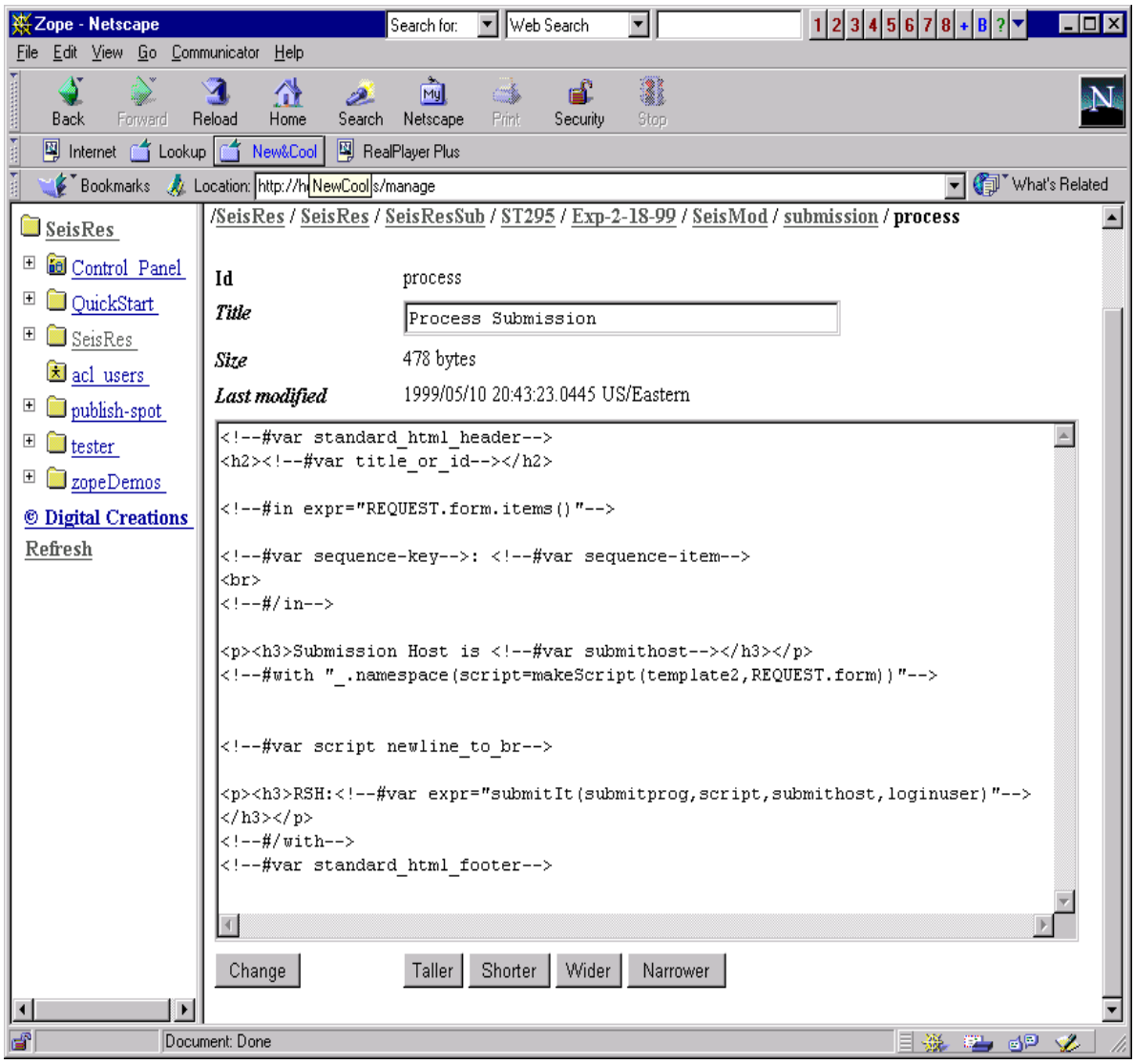


***Figure 8.*** *The DTML to build the Submission Script and execute the submission.*

Figure 9 shows the Zope authoring interface displaying the properties, including the submitprog and template properties, of the submission folder under the SeisMod folder.

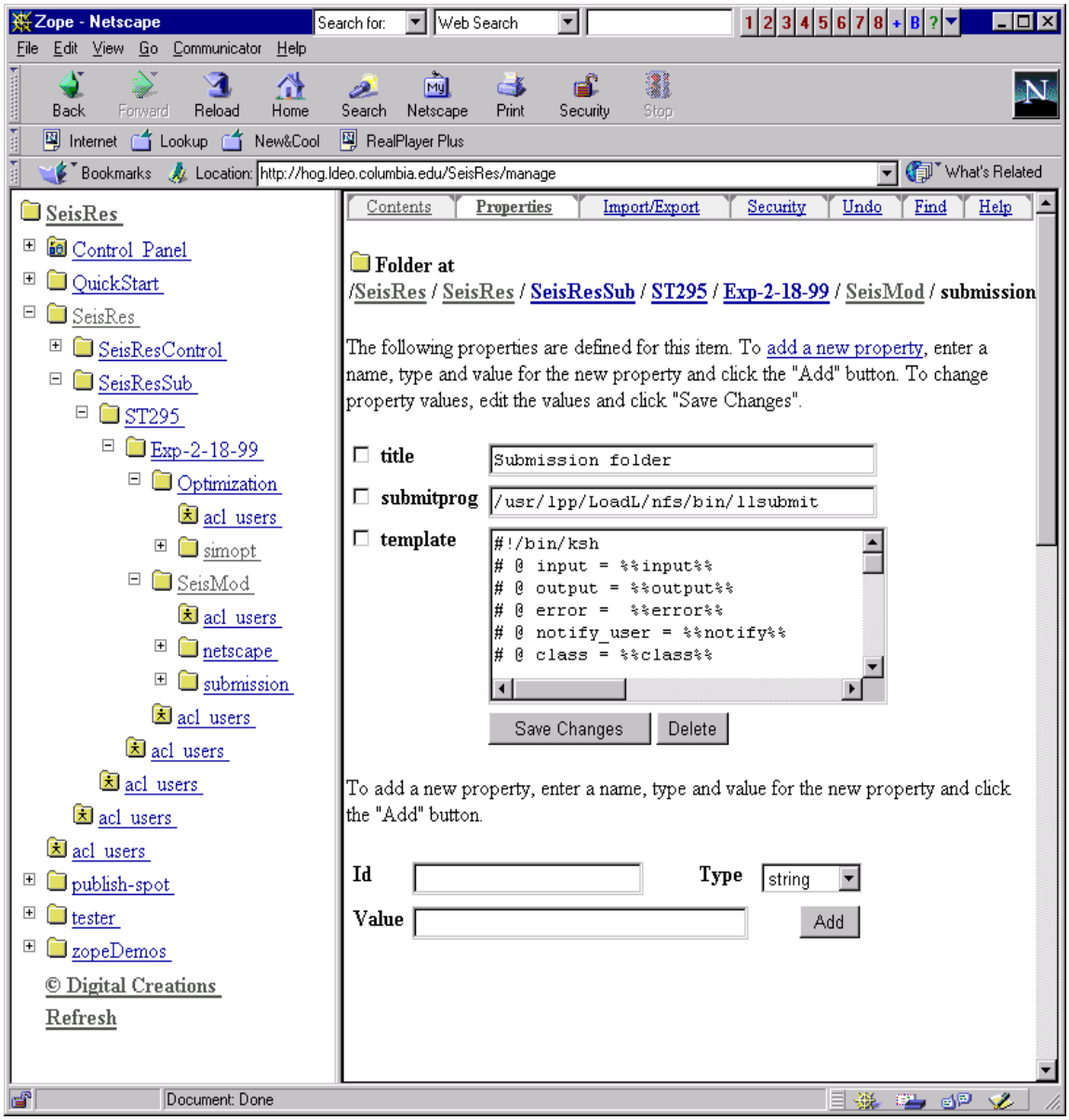


***Figure 9.*** *Properties of the Submission Folder.*

Figure 10 shows the properties of the max_processors submission variable. Note that the lastValue property holds the value of the variable from the last submission. Submission variables are implemented as empty folders with properties.

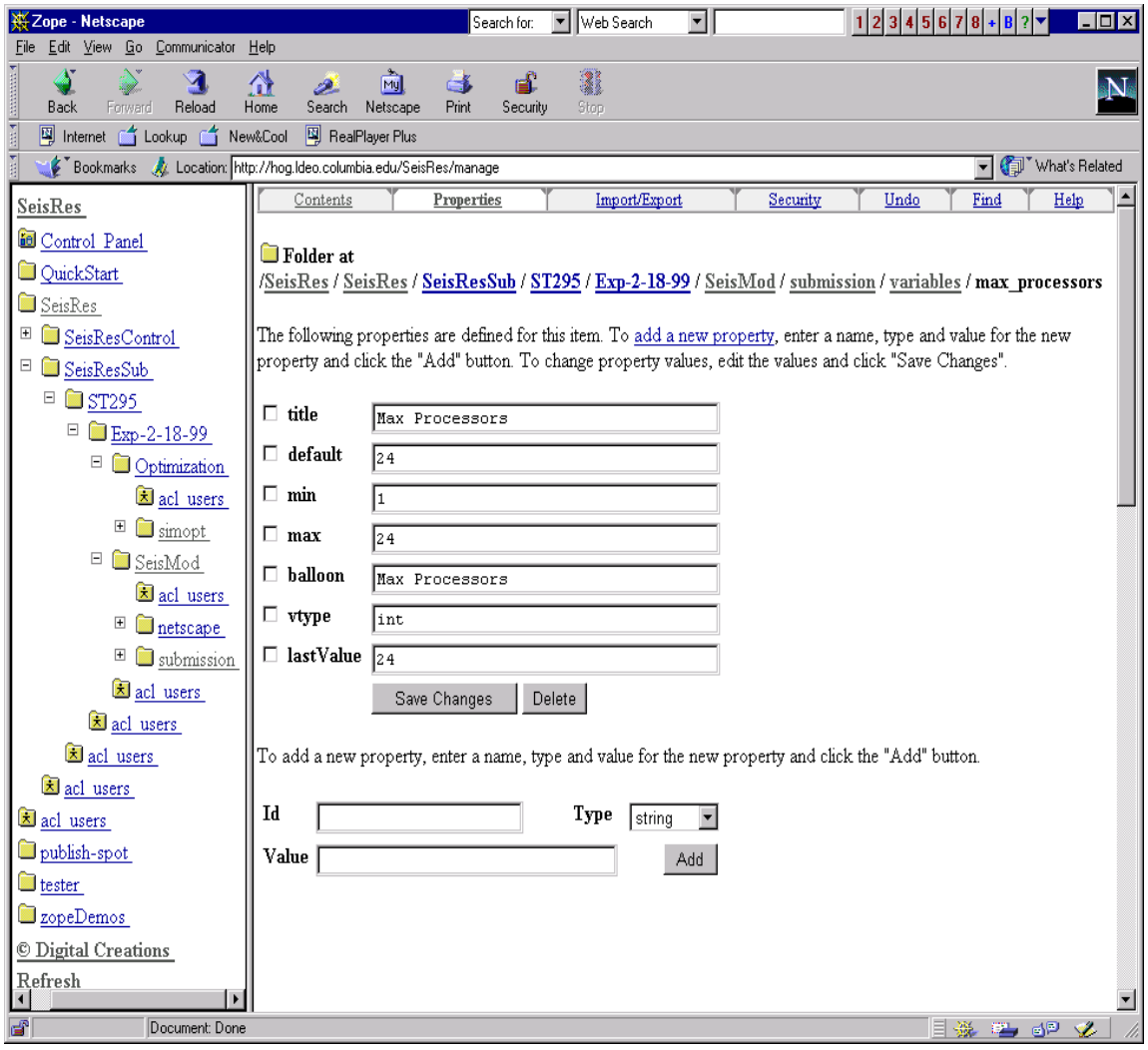


***Figure 10.*** *The properties of the max_processors variable.*

Figure 11 shows the documents under the submission folder. The subform document generates the submission form. It uses the variableFrame document and the variables in the variable folder to bulld the html (with embedded tcl for the tclet plugin) for the submisison form, Figure 6. Process is the DTML document that processes the submission and does the remote execution, Figure 7. Note that there is an extra document, Comments, that holds some deisgn comments and is not actively used. This document is an example of how Zope can be used to accumulate new associated documents pertaining to SeisRes work.

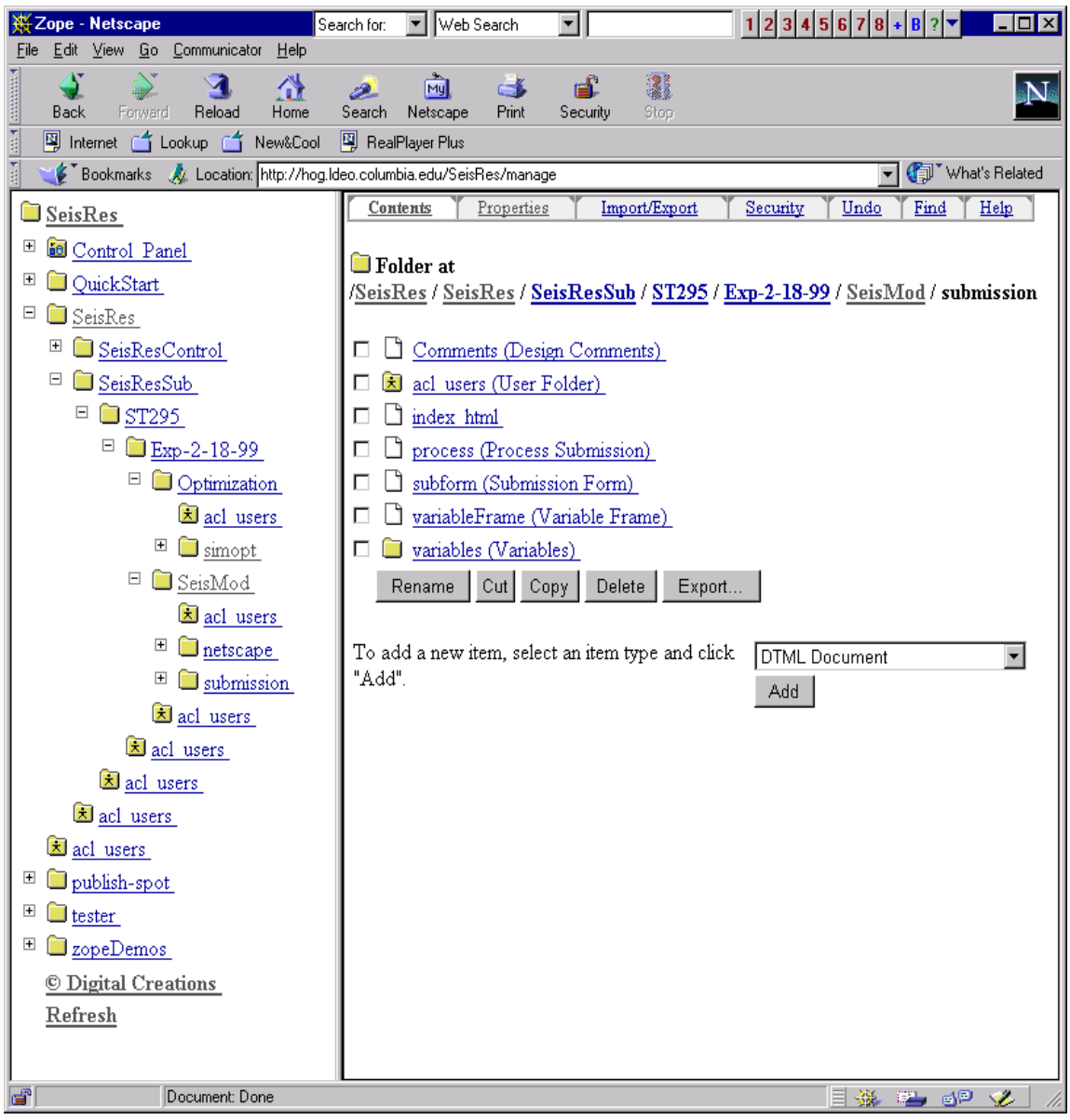


***Figure 11****. Submission Folder Documents*

We have added recently the ability to display the steps in an experiment as a workflow. The figure below is an example workflow display. This same display can also be used for dataflows.

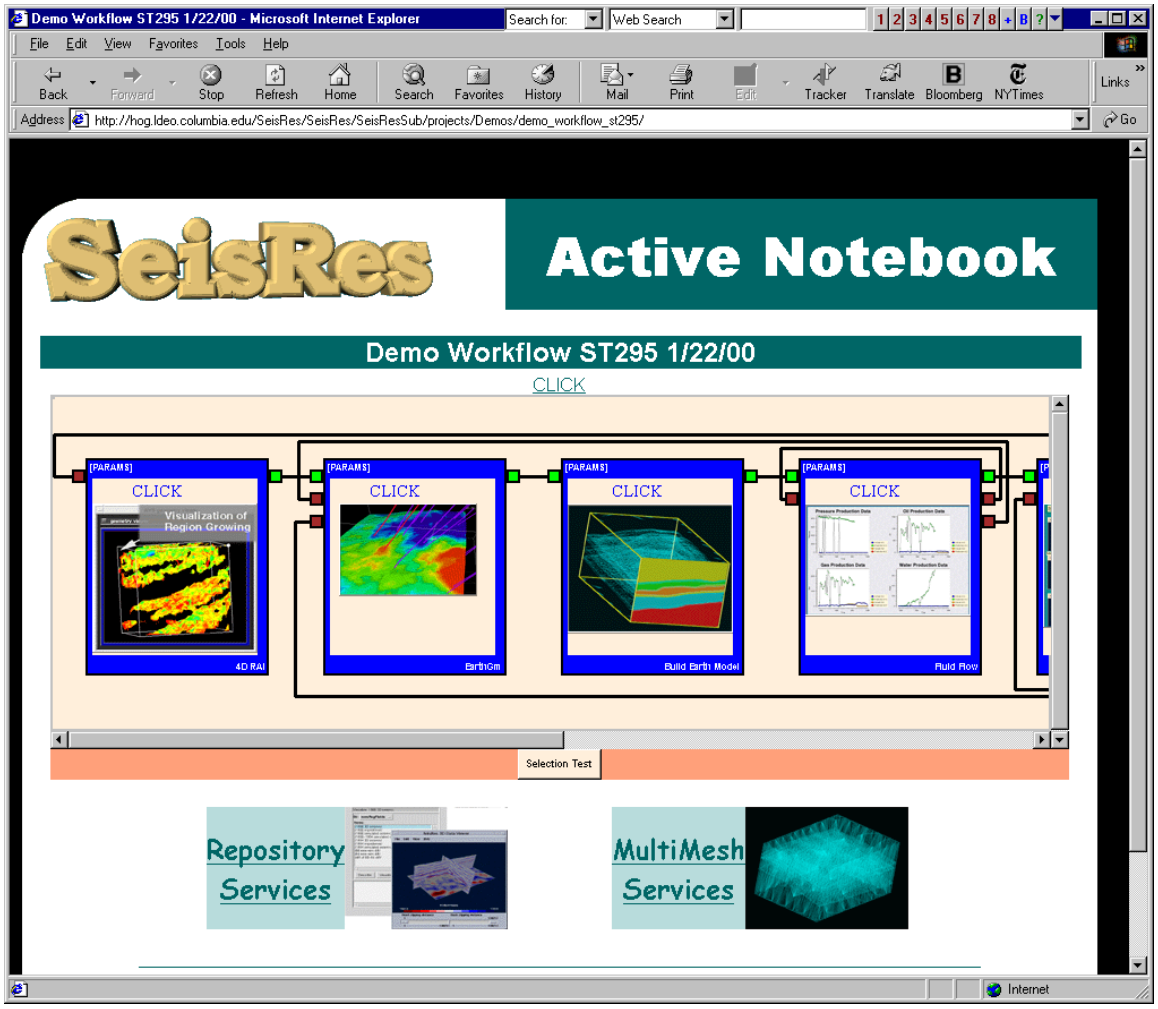


***Figure 12.*** *Workflow GUI Interface*

We now have a representation for projects and under projects are the workflows. The current organization of SeisRes contents is shown in the next figure.

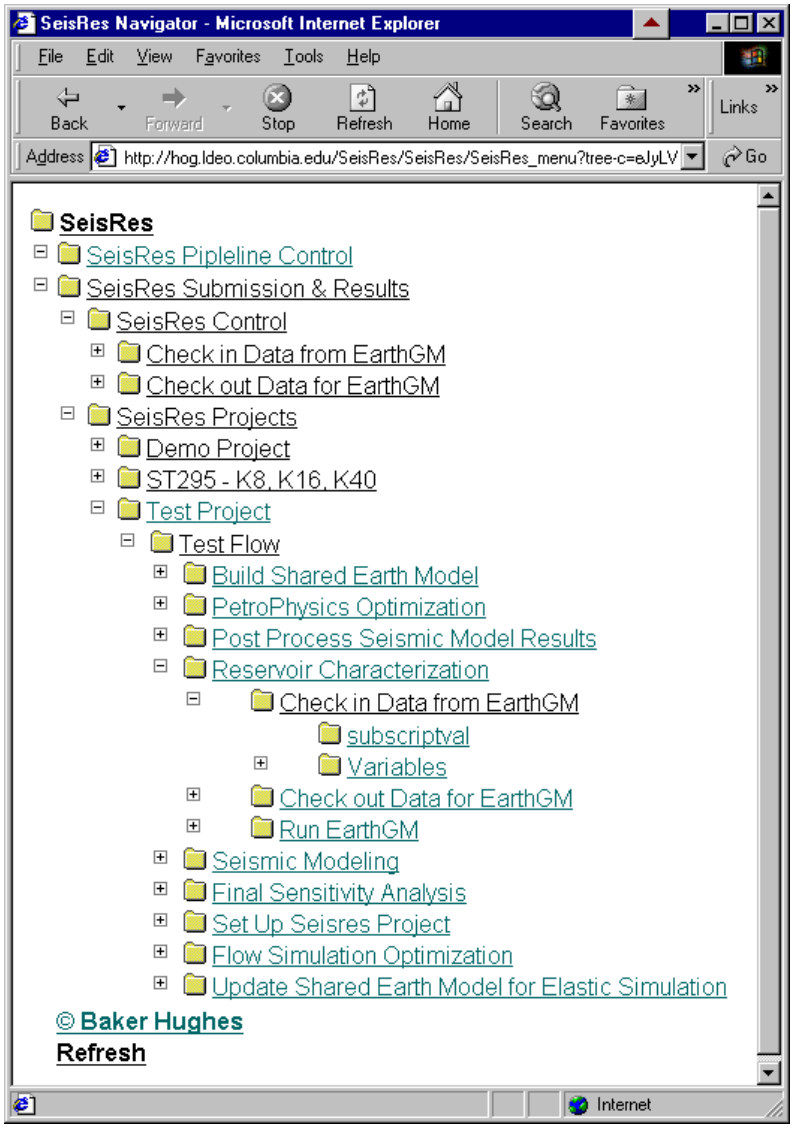


***Figure 13.*** *Current SeisRes Content*

### 2.2.7. Other Ways Of Embedding GUIs into the Notbook.

Besides the way described above, we also can embed TK interfaces into the plugin by invoking them from a tcl exec command. This allows for different versions of tcl/tk to be used from the version the plugin is based on (8.0.2). It also allows for redirecting standard output and standard error which is useful because the plugin does not handle these in a nice way (especially the diagnostic output from the c++ extensions that make up seisreswish.

The way this works is that tk frames can be a container for an external X window. We pass the window id of the tk frame as a parameter to an external wish or seisreswish invoked with exec. The example below is pio browser embedded in the notebook in this way.

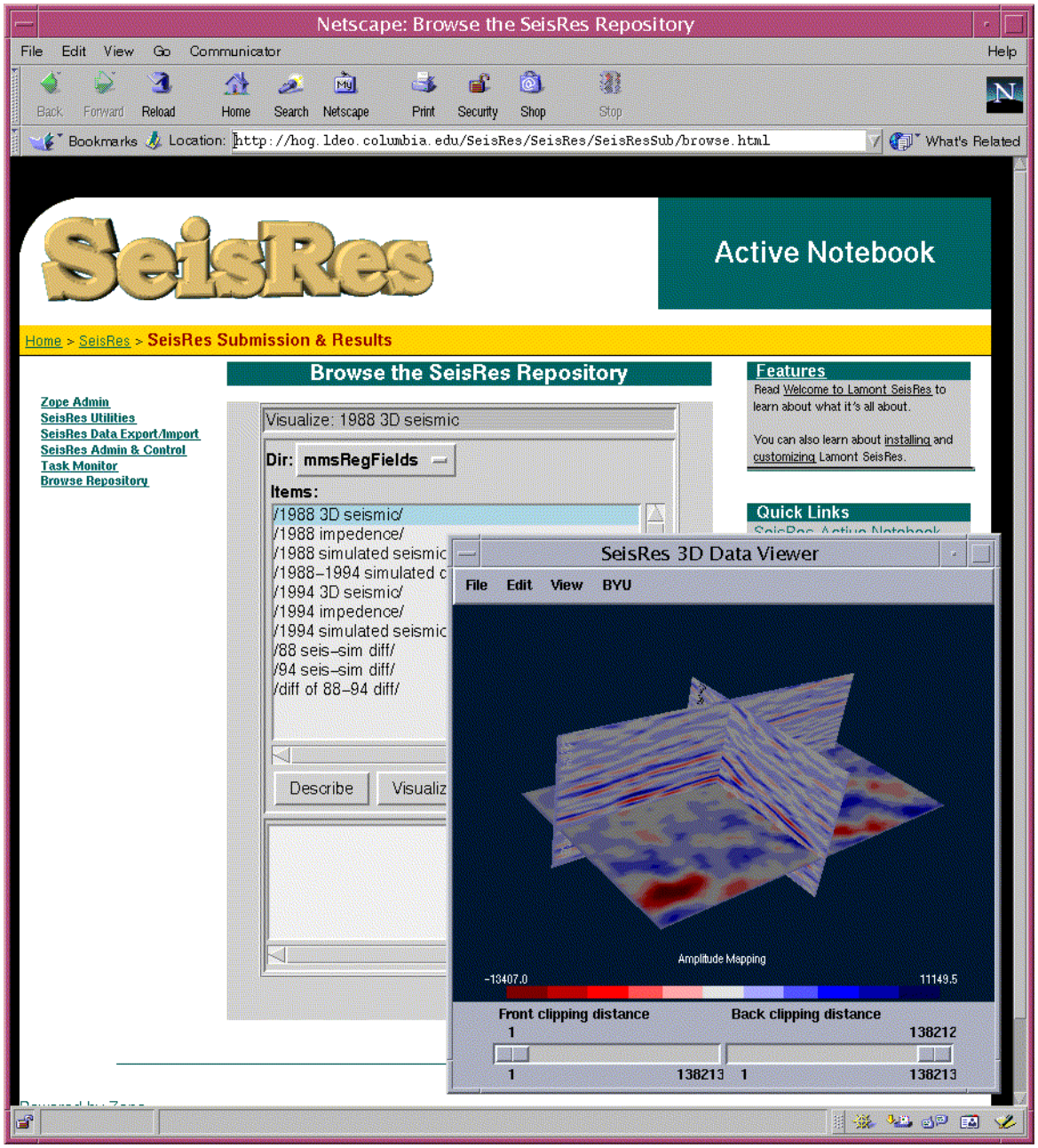


***Figure 14.*** *The PIO Browser uses a TK container frame.*

The code that invokes the browser is:

### 2. Tclet code for invoking an external tcl/tk gui.

```
<embed type="application/x-tcl" script='
catch {policy trusted}

master eval set ::env(SPEC_SERVER) 129.236.31.33
master eval set ::env(SPEC_PORT) 4096
master eval set ::env(PIO_SERVER) 129.236.31.33
master eval set ::env(PIO_PORT) 4096
master eval set ::env(EE_SERVER) 129.236.31.33
master eval set ::env(EE_PORT) 4097
master eval set ::env(SEISRES_REPOSITORY) /turf/hog/seisres
master eval set ::env(SEISRES_REPOSITORY_ROOT) /turf/hog/seisres

set ::env(SPEC_SERVER) 129.236.31.33
set ::env(SPEC_PORT) 4096
set ::env(PIO_SERVER) 129.236.31.33
set ::env(PIO_PORT) 4096
set ::env(EE_SERVER) 129.236.31.33
set ::env(EE_PORT) 4097
set ::env(SEISRES_REPOSITORY) /turf/hog/seisres
set ::env(SEISRES_REPOSITORY_ROOT) /turf/hog/seisres

source $env(SEISRES_HOME)/gui/srgui/pluginio.tcl

wm title . "SeisRes Application"
. configure -bg #009191

###This is the container
frame .01   -background #008171 -borderwidth 2 \
  -relief ridge -container true

###This is what fill the container
set res [exec $env(SEISRES_HOME)/bin/seisreswish \
  $env(SEISRES_HOME)/gui/metagui/browseinwin.tcl \
  -geometry 426x542  -use [winfo id .01]  >& /tmp/pluginstuff & ]

pack .01
after 30000  {puts [exec tail -1000 /tmp/pluginstuff] \
 ;wm iconify .pluginoutput}

' width="426" height="542">
```

The Broadway Plugin (see **www.broadwayinfo.com**) uses the X11R6 support for embedding X11 windows in web browsers. It makes uses of xrx files to describe the http post action (generally a cgi-bin script) to invoke the program whose window will appear in the area for the plugin. The cgi-scripts sets up the DISPLAY environment variable and the thin X proxy if used. An example of the xrx file for invoking EarthGM is below. Note that the `EMBEDDED` parameter will determine if the window uses the plugin's area in the browser or "floats" on the user's screen.

### 3. How to use the Broadway Plugin to invoke a X window application.

Embed for the Plugin

```
<embed src="earthgm.xrx"
      align="baseline" border="0" width="500" height="500"
      version="1.0" type="application/x-rx"></p>
```

The XRX File

```
<!-- $XConsortium: xclock /main/2 1996/10/22 14:45:12 lehors $ -->
<PARAM Name=VERSION Value=1.0>
<PARAM Name=REQUIRED-SERVICES Value=UI>
<PARAM Name=UI Value=X>
<PARAM Name=ACTION Value=http://hog.ldeo.columbia.edu/cgi-bin/xplugin.pl>
<PARAM Name=WIDTH Value=500>
<PARAM Name=HEIGHT Value=500>
<PARAM Name=EMBEDDED Value=NO>
<PARAM Name=X-UI-INPUT-METHOD VALUE=YES>
<PARAM Name=X-UI-LBX Value=NO>
<PARAM Name=X-AUTH VALUE=MIT-MAGIC-COOKIE-1>
```

The PERL SCRIPT (http post action)

```
#!/usr/local/bin/perl

# $XConsortium: Xterm /main/6 1996/12/03 20:05:55 rws $
# CGI script to launch EarthGM
#

# define every program we are going to use
#$project_root = "/usr/X11R6/bin";
$project_root = "/usr/openwin/bin";

#$command = $project_root . "/xterm";
#$command = "/src/hog/seisres/seisres/gui/runegm";
$command = "rsh medusa \" source /src/hog/seisres/earthgm.cshrc;cd
/data/hog/wei/st295/EarthGM;";

$xfindproxy = $project_root . "/xfindproxy";
```

```
$xauth = $project_root . "/xauth";

# address of our proxy manager
$proxymngr = "tcp/hog.ldeo.columbia.edu:6500";

# make stderr and stdout unbuffered so nothing get lost
select(STDERR); $| = 1;
select(STDOUT); $| = 1;

# print out header to content httpd
# no matter what happens later on
print "Content-type: text/plain\r\n\r\n";

# let's try not to leave any file behind us
# if things go really wrong
sub handler {  # 1st argument is signal name
   local($sig) = @_;
   # send error code first and error msg then
   print "1\n";
   print "Error: Caught a SIG$sig -- Oops!\n";
   system "rm -f /tmp/*$$";
   exit(0);
}

#$SIG{'INT'} = 'handler';
#$SIG{'QUIT'} = 'handler';
#$SIG{'TERM'} = 'handler';
#$SIG{'KILL'} = 'handler';
#$SIG{'STOP'} = 'handler';
# this one is perhaps the most important one,
# since this is what we should get
# when the user interrupts the GET request.
#$SIG{'PIPE'} = 'handler';


######################
# sub procedures
######################

# parse an url param of the form: proto:display[;param]
sub parse_url {
   local($input, *proto_ret, *display_ret, *param_ret) = @_;

   # extract param first
```

```
    ($sub_url, $param_ret) = split(/;/, $input, 2);
    # then extract proto and display
    ($proto_ret, $display_ret) = split(/:/, $sub_url, 2);

}

# parse an auth param of the form: auth=name:key
sub parse_auth {
    local($input, *name_ret, *key_ret) = @_;

    if ($input) {
        ($param_name, $param_value) = split(/=/, $input, 2);
        if ($param_name eq "auth") {
            ($name_ret, $key_ret) = split(/:/, $param_value, 2);
        }
    }
}

# parse an LBX param of the form: either NO or YES[;auth=...]
sub parse_lbx_param {
    local($input, *lbx_ret, *lbx_auth_name_ret, *lbx_auth_key_ret) = @_;

    ($lbx_ret, $lbxparam) = split(/;/, $input, 2);
    if ($lbx_ret eq "YES") {
        # look for an authentication auth in param
        &parse_auth($lbxparam, *lbx_auth_name_ret, *lbx_auth_key_ret);
    }
}

# setup proxy with possible auth,
# change display parameter when succeeding
sub setup_lbx_proxy {
    local(*display, $auth_name, $auth_key) = @_;

    # setup auth file for xfindproxy
    if ($auth_name && $auth_key) {
        $proxy_input = "/tmp/xlbxauth.$$";
        open(PROXYINPUT, ">$proxy_input");
        print PROXYINPUT "$auth_name\n$auth_key\n";
        close(PROXYINPUT);
        $findproxy_param = " -auth <$proxy_input";
    } else {
        $findproxy_param = "";
    }
```

```
    # remove screen number specification if there is one
    ($host, $tmp) = split(/:/, $display);
    ($dpy, $screen) = split(/\./, $tmp);
    $server = $host . ":" . $dpy;

    # let's get an LBX proxy
    open(PROXY, "$xfindproxy -manager $proxymngr -server $server -name =
LBX $findproxy_param|");
    # get the proxy address from xfindproxy output
    while (<PROXY>) {
        chop;
        ($proxy_dpy, $proxy_port) = split(/:/, $_);
        if ($proxy_dpy && $proxy_port) {
            # build up the new display name
            $display = $proxy_dpy . ":" . $proxy_port;
            if ($screen) {
                $display .= "." . $screen;
            }
            last;
        }
    }
    close(PROXY);

    if ($proxy_input) {
        system "rm -f $proxy_input";
    }
}

# add entry in authority file
sub add_auth {
    local($display, $auth_name, $auth_key) = @_;

    system "$xauth add $display $auth_name $auth_key";
}


######################
# the main thing now
######################


# handle both ways of getting query
if ($ENV{'QUERY_STRING'})
```

```
{
   $query = $ENV{'QUERY_STRING'};
} else {
   $query = $ARGV[0];
}

if ($query)
{
   $cleanup = "";

   # parse params
   %params = split(/\?/, $query);
   foreach $param (split(/\?/, $query)) {

      ($name, $value) = split(/=/, $param, 2);
      if ($name eq "WIDTH") {
         $width = $value;
      } elsif ($name eq "HEIGHT") {
         $height = $value;
      } elsif ($name eq "UI") {
# look at what we got for the UI parameter, it should be of the
# form: x11:hostname:dpynum[.screen][;auth=...]
         &parse_url($value, *proto, *display, *ui_param);
         if ($proto eq 'x11') {
            $xdisplay = $display;
         } else {
            # unknown UI protocol!!
         }
         # look for an authentication auth in param
         &parse_auth($ui_param, *xui_auth_name, *xui_auth_key);

      } elsif ($name eq "X-UI-LBX") {
         &parse_lbx_param($value, *xui_lbx,
                  *xui_lbx_auth_name, *xui_lbx_auth_key);
      }
   }

   # set authority file for X
   $ENV{'XAUTHORITY'} = "/tmp/xauth.$$";
   # and define its related cleanup command
   $cleanup = "rm -f $ENV{'XAUTHORITY'}";

   # process params
   if ($xdisplay) {
```

```
      if ($xui_lbx eq "YES") {
         &setup_lbx_proxy(*xdisplay, $xui_lbx_auth_name, $xui_lbx_auth_key);
      }
      if ($xui_auth_name && $xui_auth_key) {
         &add_auth($xdisplay, $xui_auth_name, $xui_auth_key);
      }
      # add display specification to the command line
      #$command .= " -display $xdisplay";
         $command .= " setenv DISPLAY $xdisplay; ./SCL\" &";
   }
   if ($width && $height) {
      # add geometry specification to the command line
      #$command .= " -geometry ${width}x${height}";
      # add bg & fg colors settings & active icon
      #$command .= " -bg grey -fg blue";
          #$command .= " -bg grey -fg blue +ai";

   }

# Start application followed by a cleanup command in the background.
# The ouput and input need to be redirected, otherwise the CGI process will
# be kept alive by the http server and the browser will keep waiting for
# the end of the stream...
# Catching application's failure is not easy since we want to run it in
# background and therefore we can't get its exit status. However, we can
# catch obvious errors by logging its output and after some time looking
# at whether the application is still running or not. This is determine
# based on some kind of "lock" file.
# This is quite complicated but it allows to get some error report without
# leaving any file behind us in any case.

   $LOCK= "/tmp/lock.$$";
   $LOG= "/tmp/log.$$";
   $LOG2 = "/tmp/log2.$$";

   system "(touch $LOCK; _s=true; cd /tmp; $command>$LOG 2>&1 || _s=false; if \$_s; then rm
$LOG; else rm $LOCK; fi; if [ -f $LOG2 ]; then rm $LOG2; fi; $cleanup) >/dev/null 2>&1
</dev/null";

# sleep for a while to let the application start or fail
# it's up to you to decide how long it could for the application to fail...
   sleep(5);
```

```
# now lets see if the application died
   if (open(LOCK, "<$LOCK")) {
      close(LOCK);
# the application seems to be running, remove lock and rename the log
# so that it gets removed no matter how the application exits later on
      system "rm $LOCK; mv $LOG $LOG2";
      # send success error code as reply
      print "0\n";
   } else {
# the lock file is gone, for sure the application has failed so send
# back error code and log
      print "1\n";
      system "cat $LOG; rm $LOG";
   }

} else {

   # reply with an error message
   print "This script requires to be given the proper RX arguments to run successfully.";

}
```

## 3. How to install and manage this component

### 3.1. Setting up the Server

#### 3.1.1. APACHE

If you want to use CGI-BIN scripts (there is one currently for SeisRes that is the event monitor that could be rewritten to use ZOPE with the SeiRes PYTHON interface.) or serve other pages that are not within ZOPE then one should install APACHE (or other web server like Roxen.) Otherwise, the ZOPE server could be used standalone. You can download APACHE from **http://www.apache.org** Make sure you build it with the rewrite module.
Add or modify these lines to httpd.conf:

**4. Setting up Apache**

**Set up Apache to run as the SeisRes User**

```
# User/Group: The name (or #number) of the user/group to run httpd as.
```

```
#  On SCO (ODT 3) use User nouser and Group nogroup
#  On HPUX you may not be able to use shared memory as nobody, and the
#  suggested workaround is to create a user www and use that user.
#  NOTE that some kernels refuse to setgid(Group) or semctl(IPC_SET)
#  when the value of (unsigned)Group is above 60000;
#  don't use Group nobody on these systems!

#User nobody
#Group nobody
User seisres
Group seisres
```

**Rewrite Rule for SeisRes URLs to go to ZOPE**

```
# Zope configuration maps /Zope/ to the Zope.cgi CGI script
RewriteEngine on
RewriteCond %{HTTP:Authorization}  ^(.*)
RewriteRule ^/SeisRes(.*) /src/hog/httpd/share/apache/cgi-bin/SeisRes.cgi/$1
[e=HTTP_CGI_AUTHORIZATION:%1,t=application/x-httpd-cgi,l]
```

**Enable CGI-BIN**

```
# /src/hog/httpd/share/apache/cgi-bin should be changed to whatever your ScriptAliased
# CGI directory exists, if you have that configured.

<Directory "/src/hog/httpd/share/apache/cgi-bin">
AllowOverride None
Options ExecCGI
</Directory>
```

### 3.1.2. ZOPE

- Download ZOPE from **http://www.zope.org**. Intstall Zope as someone other than root.
- Copy Zope.CGI made in $ZOPE_HOME to the cgi-bin directory for Apache and name it SeisRes.cgi.
- Add the SeisRes python extensions (makeScript.py) to $ZOPE_HOME/Extensions
- Add products (such as Zconfera) to $ZOPE_HOME/lib/python/Products .

- The suggested way to start Zope under Apache is using Zserver. An example script which is invoked from an /etc/rc2.d script that we use at Lamont is:

**5. Script to run Zope's Zserver under Apache**

```
#!/bin/csh
setenv PYTHONHOME /src/hog/zope/Zope
/src/hog/zope/Zope/bin/python /src/hog/zope/Zope/z2.py -p \
 /src/hog/httpd/share/apache/cgi-bin/Zope2.1.2.cgi
```

### 3.1.3. Client Side

Get the tcl plugin (Demailly's or NASA's) from: **http://www.demailly.com/tcl/plugin/** or **http://heasarc.gsfc.nasa.gov/Tools/maki/plugin/**

- Use Netscape (SGI's Webview, I think, should also work)
- Install the plugin using the GUI inteface provided in the install.
- Modify plugin.cfg in your ~/.netscape/tclplug/2.0/config/ (or ~/.netscape/lheatclplug/2.0/config/ for NASA's) as follows:

**6. Configuring the TCL plugin.**

```
                    Allow Trusted Policy

# The following MUST not be allowed unless high trust is granted:

#    disallow trusted
allow trusted
```

```
                Allow a large number of Netscape frames
# This part allow the administrator/user to limit the total number
# of frames all the tclets can access :

variable maxFrames 10000
```

Set your .hosts to allow seisres to to a rsh for you. (In the future, we will be using ssh so that would have to be configured.) For example, in my .rhosts file I have:

```
dragon.ldgo.columbia.edu seisres
hog.ldgo.columbia.edu seisres
rat.ldgo.columbia.edu seisres
ox.ldgo.columbia.edu seisres
horse.ldgo.columbia.edu seisres
medusa.ldgo.columbia.edu seisres
```

Set the following environment variables pertaining to the notebook:

### 7. Shell environment variables for the notebook.

| VARIABLE | VALUE | OPTIONS |
|---|---|---|
| TIXHOME | /usr/local/lib/tix4.1 | Typically. If you use TIX in plugin. |
| TCL_PLUGIN_WISH | 1 | Needs to be set to 1 instead of default 0 |
| SEISRESWISH_CLIENT_FLAG | 1 | Governs how the pioChoosers are run.<br>Can have the following values:<br>0 - Run inside the current process.<br>1 - Launch, using exec. (The default value.)<br>2 - Run in the separate "seisreswish-client". |
| SEISRESWISH_SERVER_PORT | 9876 | If SEISRESWISH_CLIENT_FLAG is set to 2 |
| SEISRESWISH_CLIENT_HOST | Ip of host | If SEISRESWISH_CLIENT_FLAG is set to 2 |

In addition, if you use TIX in the plugin, you need to make TIX libs searchable by the plugin. One way to do this is to make a directory called tix4.1 in ~/.netscape/tclplug/ if you use the tclplugin (or ~/.netscape/lheatclplug/ if you use the NASA plugin) and add a link there called library linked to typically /usr/local/lib/tix4.1.

## 4. Important algorithms in this component

There are none.

## 5. References to other documentation (http links and paper docs)

- **http://www.kitware.com/vtk.html** VTK
- **http://www.demailly.com/tcl/plugin/** tcl web plugin
- **http://www.cacr.caltech.edu/isda/report.html** Workshop on Interfaces to Scientific Data Archives Pasadena, California 1998 March 25-27
- **http://ala.vsms.nottingham.ac.uk/vsms/java/epub/xml.html** An introduction to Structured Documents
- **http://www.oasis-open.org/cover/xml.html** The SGML/XML Web Page
- **http://www.oasis-open.org/cover/xml-data9706223.html** XML-Data
- **http://home.worldcom.ch/~jmlugrin/fesi/index.html** FESI EcmaScript Interpreter in Java
- **http://www.python.org/jpython/whatis.html** Python interpreter in JAVA

- **http://developer.netscape.com/docs/technote/javascript/liveconnect/liveconnect_rh.html** LiveConnect
- **http://developer.netscape.com/docs/manuals/enterprise/wrijsap/lc.htm** Server-Side LiveConnect
- **http://www.webcoder.com/howto/15/index.html** DOM Tutorial
- **http://www.w3.org/TR/REC-DOM-Level-1/** W3C Document Object Model (DOM) Level 1 Specification
- **http://www.zope.org/** Zope
- **http://www.fmi.uni-passau.de/Graphlet/** tcl-based Graphlet
- **http://www.stratasys.com/software/metagui/** MetaGui Package

### 6. How do you create and edit the configure.in file.  Provide example code.

Not relevant for the Notebook.

### 7. Servers PIO, EE, apache.  Associated environment variables required.

See section 3.1.3 for a list of environment variables for the client.
Settings in notebook.  These are setting to be used for scripts executed by the server. They are also used to set the PIO & EE servers and the SEISRES_REPOSITORY for the client.

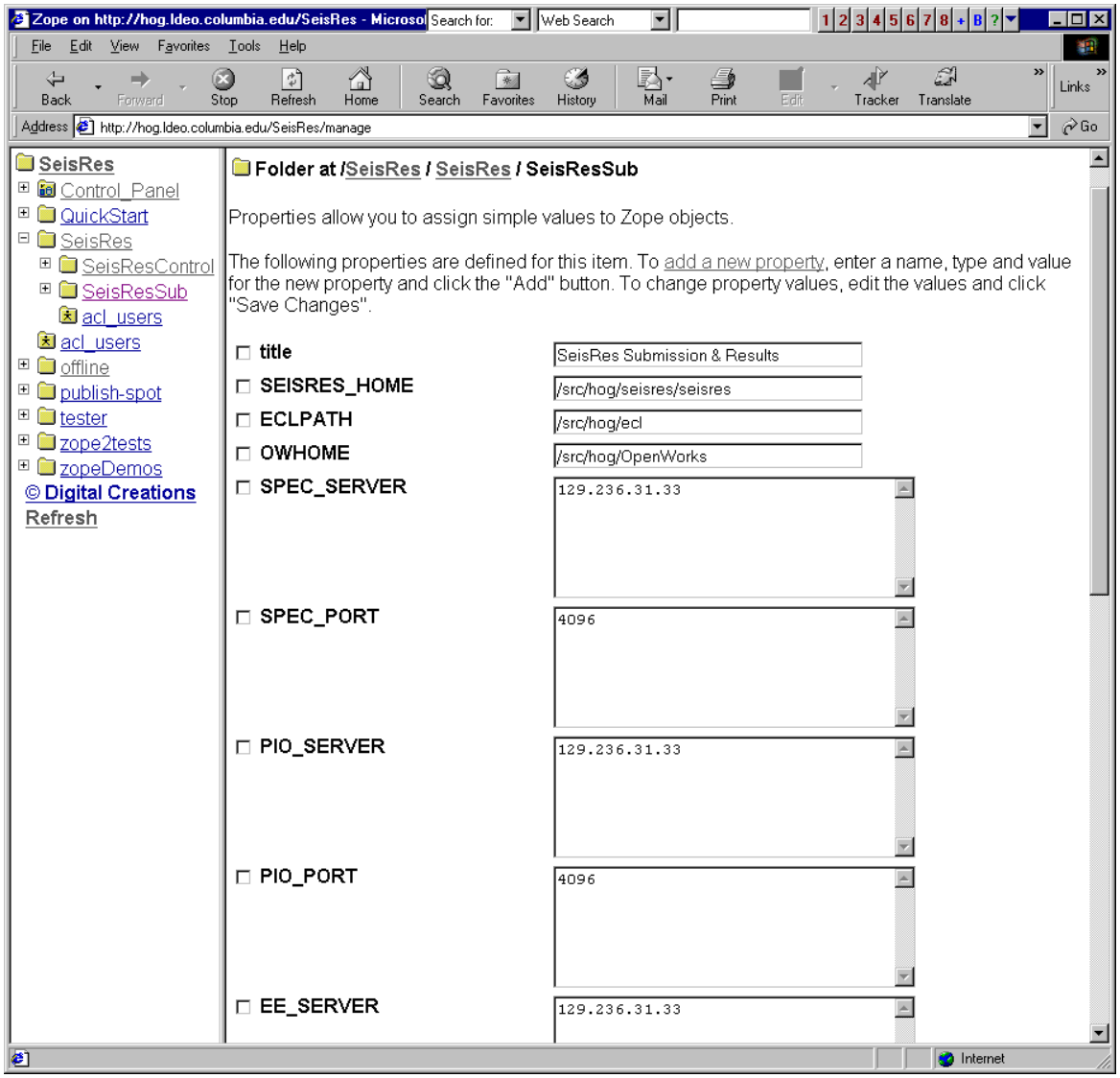


***Figure 15**. Environment variables set in the notebook*

The full list currently is:

### 8. List of environment variables set in notebook.

| | |
|---|---|
| **SEISRES_HOME** | /src/hog/seisres/seisres |
| **ECLPATH** | /src/hog/ecl |
| **OWHOME** | /src/hog/OpenWorks |
| **SPEC_SERVER** | 129.236.31.33 |
| **SPEC_PORT** | 4096 |
| **PIO_SERVER** | 129.236.31.33 |
| **PIO_PORT** | 4096 |
| **EE_SERVER** | 129.236.31.33 |
| **EE_PORT** | 4097 |

| | |
|---|---|
| **SEISRES_REPOSITORY** | /turf/hog/seisres |
| **PATH** | .:/usr/bin:/usr/ucb:/usr/dt/bin:/usr/openwin/bin:/src/hog/usr/local/bin:/usr/local/bin:%%SEISRES_HOME%% |
| **LD_LIBRARY_PATH** | /usr/local/SUNWspro/SC4.0/lib:/usr/openwin/lib:/src/hog/usr/local/lib:%%SEISRES_HOME%%/lib:/src/hog/usr/local/lib:%%SEISRES_HOME%%/ext/vtk/lib |
| **submitprog** | /src/hog/usr/local/bin/tclsh |
| **SEISRES_REPOSITORY_ROOT** | /turf/hog/seisres |