CS 535 Object-Oriented Programming & Design
Fall Semester, 2003
Doc 17 WebToolkit

References

VisualWorks Web Application Developer's Guide, WebAppDevGuide.pdf in the VisualWorks documentation folder

Useful VisualWorks Extras

AutoComplete

After loaded the tab key will activate

Attempts to auto complete name of class, method, and variable you currently are typing

Load parcel in Environment Enhancements in Parcel Manager

RB_Tabs

Adds tab in browser for each view you create

Minor bug – when you restart VisualWorks tabs are not shown until you select a view

Load parcel in Environment Enhancements in Parcel Manager

AddInstVarAtCompile

When you save/compile code in browser that contains a new name, the pop window will allow you to define the name as a instance variable or class in addition to the standard options

Download from the class web site

VisualWorks Web Server

Supports:
- Static web pages
- Servlets (Java spec 2.2)
- Smalltalk server pages (ASP 3.0, JSP 1.1)
- HTTPS
- CGI, ISAPI, FastCGI
- Cookies
- Session

Can be run:
- Standalone
- Behind Apache or IIS

Servlets Verses Server Pages

Systems for dynamic web pages

Servlets
- All Smalltalk code
- Allow developer to work in normal environment
- Requires code to generate html

Server Pages
- Can use Html tool to generate html
- Embed Smalltalk in html pages
- Harder to debug

VisualWorks WebToolkit supports both

We will use servlets

Configuration Files

Copy the files listed below from VisualWorks installation /web to the directory containing your image

configure-site.ini
default-site.ini
webtools.ini

These files help configure the VW web server

In default-site.ini edit the line

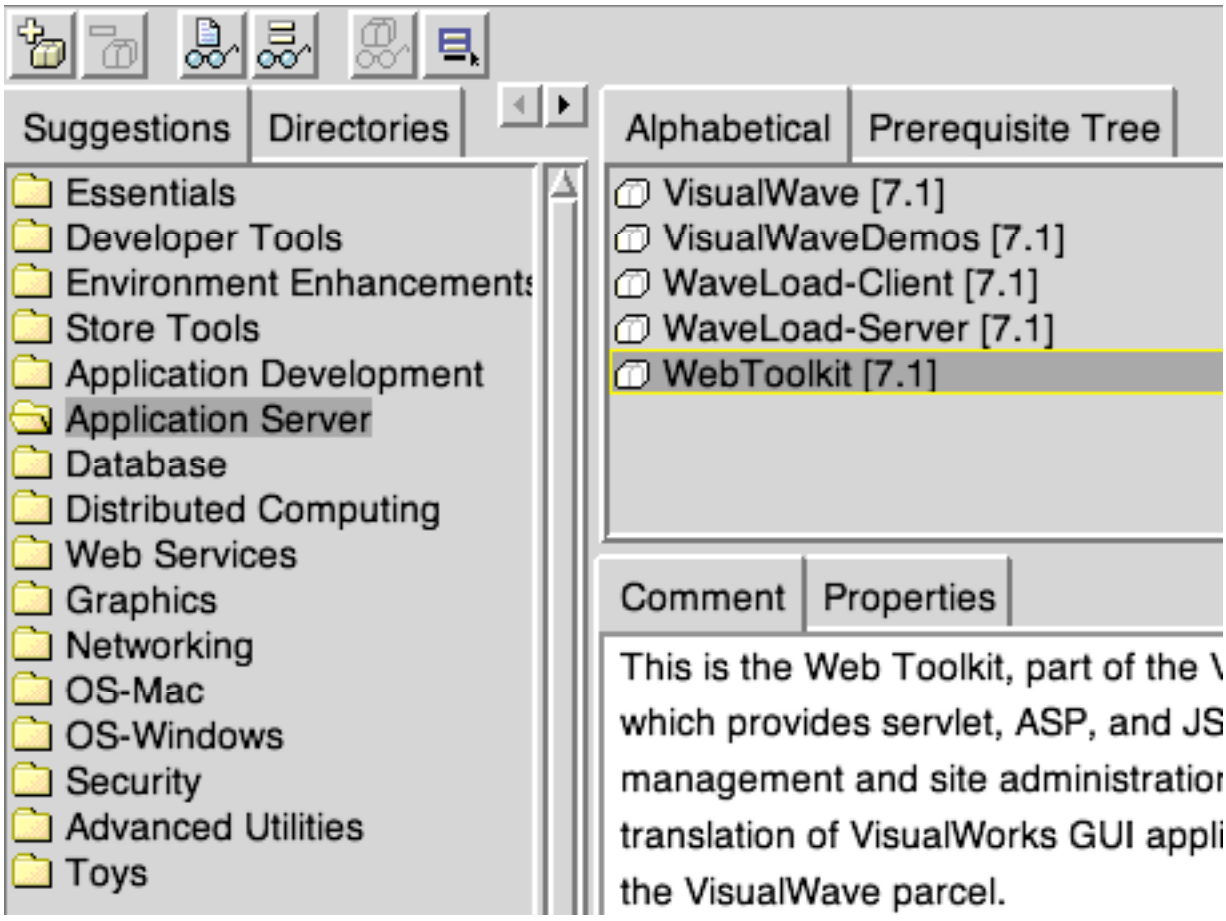
```
directory = $(VISUALWORKS)/web/examples
```

so that directory is set to the location you want the web server to look for web pages

Loading WebToolkit

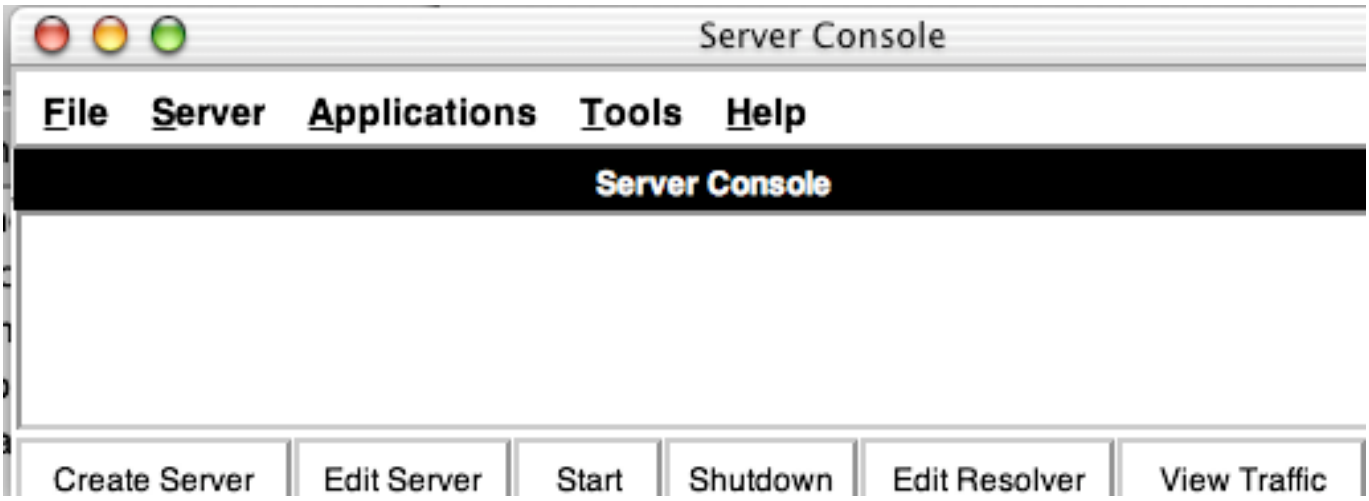
First open the Parcel Manager by selecting “Parcel Manager” item in the “System” menu on the Launcher. Or you can click on the “Parcel Manager” icon  on the Launcher.

In the “Parcel Manager” select “Application Sever” in the left pane. In the right pane select the “WebToolkit” parcel. To load the parcel select the “Load” item in the “Parcel” menu in the “Parcel Manager”.

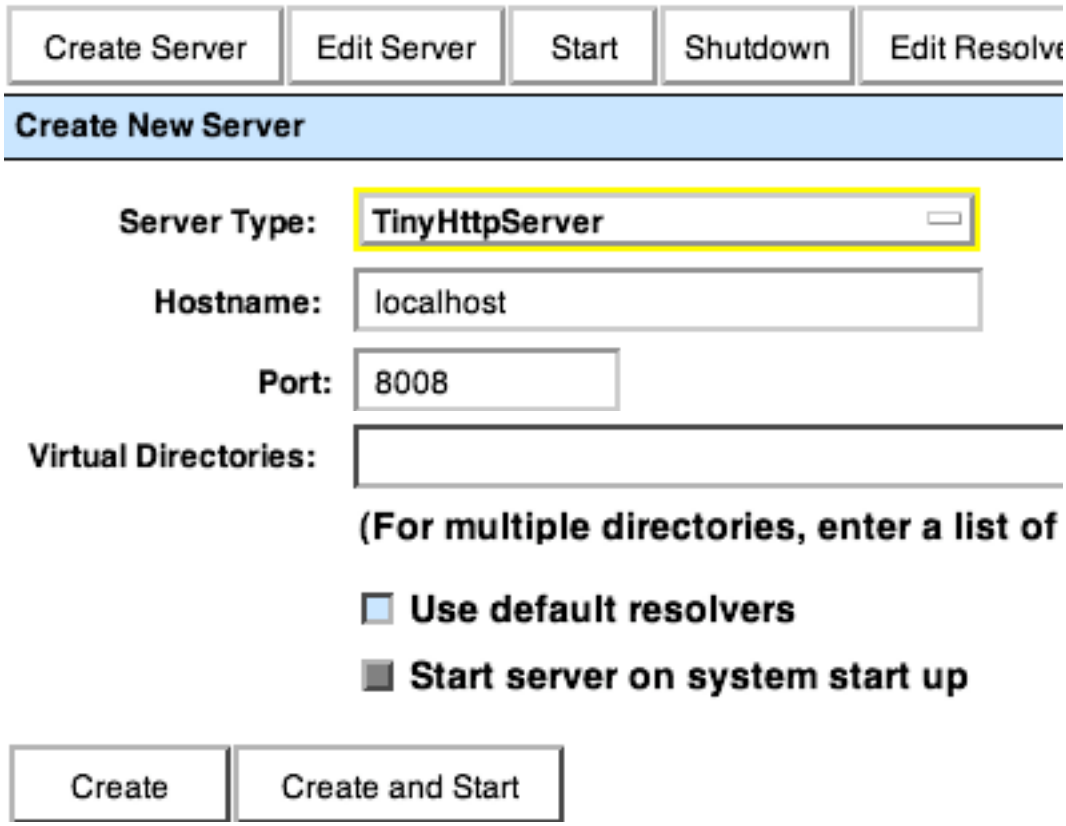


Starting Web Server

To start the VW web server you need to use the “Server Console”. After loading the WebToolKit select the “Server Console” item of the “Web” menu item on the Launcher. The Web menu was added by the WebToolkit. One can also open the “Server Console” by clicking on the “Server Console” icon  in the Launcher.



In the Server Console, click on the “Create Server” button. A pane will appear at the bottom of Server Console window. Set the “Server Type” to “TinyHttpServer”. You may use any port you wish. However, some platforms require root or administrator privileges to use ports under 1024. I will use the default port of 8008. Click on the “Create and Start” button to create and start the server.



Testing the Server

Using a browser on the same machine access the url:

http://localhost:8008/echo

This should list the headers sent to the web server. On some machines you may need to use:

http://127.0.0.1:8008/echo

The address 127.0.0.1 is the local machine. Other urls to try are given below. From now on I will use 127.0.0.1 and localhost interchangeably.

http://127.0.0.1:8008/servlet/VeryBasicServlet

The first url is an example of a servlet in Smalltalk. The second provides some examples of Smalltalk server pages.

Some Useful URLS

http://localhost:8008/configure/
A page that allows you to manage and configure the web server

Creating A Servlet

VW allows you to create servlets, using an API similar to Java's servlets. To create a new servlet create a subclass of VisualWave.HttpServlet. Note that you need to type the full name of the class, including the VisualWave namespace. Here is a simple servlet.

```
Smalltalk defineClass: #HelloWorld
   superclass: #{VisualWave.HttpServlet}
   indexedType: #none
   private: false
   instanceVariableNames: ''
   classInstanceVariableNames: ''
   imports: ''
   category: 'CS535'!

!HelloWorld methodsFor: 'servering'!

doGet: aRequest response: aResponse
   aResponse write:
      '<HTML><BODY>Hello world</BODY></HTML>'! !
```

Once you add this to your image you can access the servlet using the url:

http://127.0.0.1:8008/servlet/HelloWorld

The basic format is

http://serverAddress/servlet/ClassName

Important Servlet Methods

initialize
doGet:response:
doPost:response:

initialize is called when an instance of the servlet class is called

doGet:response:

- Called when an http get request is received
- Normal non-form requests use this

doPost:response:

- Called when an http post request is received
- Used for forms

Response

doGet: aRequest response: aResponse

aResponse is an instance of VisualWave.Response

Useful methods

write: anObject

Writes a string representation of anObject to the output stream

Contents of the output stream are sent back to the web browser

cr

Adds a cr to the output stream

space

Adds a space to the output stream

redirectTo: aUri

Sends a redirect back to the web browser to the uri given in string form

Form Example

Form

```
<HTML>
   <title>Create a Survey</title>
<BODY>
<p>
<center>CS 535 Surveys </center>
<HR size=1 noshade>
<h2 align="center">Add a new Survey</h2>
<form action="http://127.0.0.1:8008/servlet/NewSurvey"
method="post">
<textarea name="definition" rows="20" cols="80">
</textarea>
<BR>
<input type="submit"></input>
</form>
</BODY></HTML>
```

Servlet

```
Smalltalk defineClass: #NewSurvey
   superclass: #{VisualWave.HttpServlet}
   indexedType: #none
   private: false
   instanceVariableNames: ''
   classInstanceVariableNames: ''
   imports: ''
   category: 'CS535'

doPost: aRequest response: aResponse
   | surveyString |
   surveyString := aRequest anyFormValueAt: 'definition'.
   "Here we can do anything we want.
   Just return it for demonstration"
   aResponse
      write: '<html><body>You entered: ';
      write: surveyString;
      write: '</body></html>'
```

Connections

```
<form action="http://127.0.0.1:8008/servlet/NewSurvey"
method="post">
```

Html form action contains the url to the servlet

Uses post method

doPost: aRequest response: aResponse

Servlet implements doPost:response: method

```
<textarea name="definition" rows="20" cols="80">
</textarea>
```

Textarea name is “definition”

```
surveyString := aRequest anyFormValueAt: 'definition'.
```

Use the name of the textarea to get the value form the request

Request & Forms

VisualWave.Request>>allFormValuesAt: aString

"Return all the form values associated with aString"

VisualWave.Request>>anyFormValueAt: aString

"Return the first form value associated with aString. If there are none, return nil"

VisualWave.Request>>form

"Return a dictionary of all form data, with the name of the data as a key, and a collection of all values by that name as the value"

Query Data

Forms sent using get rather than post

Url format:

http://127.0.0.1:8008/foo/bar?name=roger&office=561

VisualWave.Request>>allQueryValuesAt: aString

"Return all the query data associated with aString"

VisualWave.Request>>anyQueryValueAt: aString

"Return the first query data associated with aString. If there are none, return nil"

VisualWave.Request>>query

"Return the query data associated with this request.
This is a dictionary, mapping from the name of a query parameter to the collection of values associated with it."

Example

```
Smalltalk defineClass: #Information
   superclass: #{VisualWave.HttpServlet}
   indexedType: #none
   private: false
   instanceVariableNames: ''
   classInstanceVariableNames: ''
   imports: ''
   category: 'CS535'
```

Instance Method

```
doGet: aRequest response: aResponse
   | answer userRequest |
   userRequest := aRequest anyQueryValueAt: 'type'.
   answer := 'No valid request given'.
   userRequest = 'time' ifTrue:[answer := Time now printString].
   userRequest = 'date' ifTrue:[answer := Date today printString].
   aResponse
      write: '<html><body>Answer: ';
      write: answer;
      write: '</body></html>'
```

Example URLs

http://127.0.0.1:8008/servlet/Information?type=date
http://127.0.0.1:8008/servlet/Information?type=time

Form & Query Data

The following method work with post and get data

VisualWave.Request>>allParameterValuesAt: aString

VisualWave.Request>>anyParameterValueAt: aString

VisualWave.Request>>parameters

Unused Path Information

```
Smalltalk defineClass: #Information
   superclass: #{VisualWave.HttpServlet}
   indexedType: #none
   private: false
   instanceVariableNames: ''
   classInstanceVariableNames: ''
   imports: ''
   category: 'CS535'
```

Instance Method

```
doGet: aRequest response: aResponse
   | pathString pathComponents answer |
   pathString := aRequest pathInfo.
   pathComponents := pathString tokensBasedOn: $/.
   answer := 'No valid request given'.
   pathComponents last = 'time'
      ifTrue:[answer := Time now printString].
   pathComponents last = 'date'
      ifTrue:[answer := Date today printString].
   aResponse
      write: '<html><body>Answer: ';
      write: answer;
      write: '</body></html>
```

Example URLs

http://127.0.0.1:8008/servlet/Information/time
http://127.0.0.1:8008/servlet/Information/date

Servlet Debugging

There is no need to stop & start the server to:

- Add new servlets
- Modify servlets
- Debug servlets

You can

- Place “self halt” statements servlet methods
- Request the servlet from a Web browser
- Enter the debugger
- Step through the servlet code
- Modify the servlet code
- And still return a response to the Web browser

Templates

Generating html in servlets can be difficult

Often much of the html is static

- Header
- Footer

Store static html in a file

Add placeholders in file for html to be generated dynamically

Template class can be download from the course web site

Example

@@@ indicates start/end of placeholder
Can have different place holders in one file
Same placeholder can occur in multiple places

Surveys.html file

```
<BODY>
<p>
<center>
   CS 535 Surveys <br>
   Fall Semester, 2003
</center>
<HR size=1 noshade>
@@@body@@@
</BODY>
</HTML>
```

Using the Template

```
Smalltalk defineClass: #HelloWorld
   superclass: #{VisualWave.HttpServlet}
   indexedType: #none
   private: false
   instanceVariableNames: ''
   classInstanceVariableNames: 'template '
   imports: ''
   category: 'CS55Servlets'
```

Instance Methods

```
doGet: aRequest response: aResponse
   | template |
   template := Template fromFile: 'hello.html'.
   template at: 'body' put: 'Hello World'.
   aResponse write: template asString.
```