

## Chapter 32

# Water App: Building Applications with Water

### IN THIS CHAPTER

- ◆ Creating a Web App with `defclass`
- ◆ Creating a dynamic page with `defmethod`
- ◆ Understanding how an HTML form represents a method call
- ◆ Using the data from a submitted HTML form

**WATER APP DESCRIBES HOW TO USE WATER** for building Web-based applications. Web-based applications use HTML for user display and input, and a Web browser for presentation. Water App is the application framework for Water, and is comparable to JSP and ASP.

## Creating a Simple Application

The following example creates a simple Water App. A `defclass` is used to define the application and the name of the class is the name of the application. `my_app` contains a single page named `start`.

```
<defclass my_app>
  <defmethod start>
    <H1>Starting Page</>
  </defmethod>
</defclass>
```

### Calling the start page

By convention, when a class is displayed as HTML, it looks for a `start` method and calls it. In this case, the implementation of `start` is a single `H1` hypertext tag with a content of `Starting Page`. The following example requests `my_app` that outputs the starting page.

```
my_app → <H1>Starting Page</H1>
```

Alternatively, you can call `start` directly.

```
my_app.<start/> → <H1>Starting Page</H1>
```

## Creating an Application Server

To make the application available through a Web browser, you need to create an application server. To create an application server, call `server` and give it the object to serve. In this case, the object to serve is the new application named `my_app`.

```
<server my_app/>
```



Chapter 14 on Water Server describes how to create custom servers.

---

The preceding example creates a new `server` on port 80 that serves the `my_app` application as the `root_object`. You can access the new application through a standard Web browser. Requesting the following URI will return the starting page of the application.

```
http://localhost  
→ <H1>Starting Page</H1>
```

## Extending a Simple Application

The `start` method could perform some calculations to determine which page to show. The following example shows the `afternoon_page` if it is after noon; otherwise, it shows the `morning_page`.

```
<defclass my_app>  
  <defmethod start>  
    <if> time.current.pm  
      .<afternoon_page/>  
    else  
      .<morning_page/>  
    </if>  
  </defmethod>  
  
  <defmethod morning_page>  
    <H1>Morning page</>  
  </defmethod>
```

```
<defmethod afternoon_page>
  <H1>Afternoon page</>
</defmethod>

</defclass>
```

## Multiple pages

The following example adds another page to the application:

```
my_app.
  <set second_page=
    <H3>This is the second page</>
  />
```

Alternatively, you could have redefined `my_app` with a second page.

```
<defclass my_app
  second_page=<H3>This is the second page</>
>
  <defmethod start>
    <H1>Starting Page</>
  </defmethod>
</defclass>
```

### How Does the Starting Page Get Returned?

If the URI/URL has no path, it returns `root_object` for the server, which is `my_app` in this example. The `to_html` method gets called on `my_app` before it is returned from the server. The `to_html` method, in turn, calls the `start` method on the `root_object`.

```
my_app.
  <defmethod to_html>
    .<start/>.<to_html/>
  </defmethod>
```

The `start` method must return an instance of hypertext. Calling `to_html` on that instance returns an HTML page to be displayed in the browser.

```
my_app.<to_html/> → <H1>Starting Page</H1>
```

The `to_html` method on a class calls `start` by default, and the output above is the result of calling the `start` method.

The second page can be called through the URI/URL in a browser.

```
http://localhost/second_page
→ <H3>This is the second page</>
```

Suppose you redefine the start method to link to the second\_page.

```
my_app.
<defmethod start>
  <H1>This is the starting page.<BR/>
  <A href=<uri second_page/> >Click here for page 2.</A>
</H1>
</defmethod>
```

Now use a browser to view the starting page.

```
http://localhost/ →
<H1>This is the starting page.<BR/>
  <A href="/second_page" >Click here for page 2.</A>
</H1>
```

Clicking the link brings you to the second page.

An XHTML page can be loaded from the filesystem or the Web and assigned to a page field in my\_app. This makes it possible to view static HTML pages in a browser and still use those pages in your application. The following example sets the second\_page field to a string containing the contents from a Web page.

```
my_app.<set
  second_page=web.<web "http://www.waterlang.org"/>.content
/>
```



Chapter 12 on Water Web describes a web resource.

---

## Adding a page method

An application is composed of static pages as well as dynamic pages that take arguments. Water App uses a Water method for dynamic pages, and the method is defined with defmethod.

```
my_app.
<defmethod a_dynamic_page a_color="red" username="plusch">
```

```
<H1 color=a_color>
  This is a dynamic page for username: <do username/>
</H1>
</defmethod>
```



Chapter 5 on Water Contract describes how to create contracts using `defmethod`.

`a_dynamic_page` can be called through a Web browser. Because the method has default values for all the arguments, the following page is returned:

```
http://localhost/a_dynamic_page? →
<H1 color="red">
  This is a dynamic page for username: plusch
</H1>
```

The method can also be called with arguments passed in the URI/URL.

```
http://localhost/a_dynamic_page?a_color=blue&username=angela
→
<H1 color="blue">
  This is a dynamic page for username: angela
</H1>
```

The method could also be called directly from Water.

```
my_app.<a_dynamic_page "blue" "angela"/> →
<H1 color="blue">
  This is a dynamic page for username: angela
</H1>
```

## Calling a dynamic page with an HTML form

A dynamic page is usually called by submitting an HTML form. An HTML form can be thought of as filling in the arguments to a method call. Pressing the submit button on the form calls the method.

```
my_app.<set second_page=
  <FORM action=<uri a_dynamic_page/> >
  <H1>Second page</>
  Color: <INPUT name="color"/> <BR/>
  Username: <INPUT name="username"/> <BR/>
```

```
<INPUT type="submit" value="Submit"/>
</FORM>
/>
```

Show the form in a browser by typing in the following URI/URL:

```
http://localhost/second_page
```

If you fill in the fields with values "purple" and "fry", and then press Submit, the following page is returned:

```
<H1 color="purple">
  This is a dynamic page for username: fry
</H1>
```

All the basics of Water App have been covered. Every other feature of Water App is extending the previous design patterns.

## Subapplications

A Water App is implemented with `defclass`. A Water App can contain subapplications by defining subclasses.

```
<defclass my_app>
  <defmethod start>
    <HTML>
      <H1>Starting Page for my_app</>
      <A href=<uri subapp/>>Start subapp</A>
    </HTML>
  </defmethod>

  <defclass subapp>
    <defmethod start>
      <H1>Starting Page for subapp</>
    </defmethod>
  </defclass>
</defclass>
```

You can launch a subapp from a URI/URL by clicking a link or typing the following URI/URL directly:

```
http://localhost/subapp
```

Alternatively, you can start the application from Water.

```
my_app.subapp →
<H1>Starting Page for subapp</>
```

The same subapplication could be added to multiple applications under different paths. The next example defines an application named `subapp_A`. `my_app` includes `subapp_A` by setting its `subapp` field to `subapp_A`.

```
<defclass subapp_A>
  <defmethod start>
    <H1>Starting Page for subapp_A</>
  </defmethod>
</defclass>

<defclass my_app
  subapp=subapp_A
>
  <defmethod start>
    <HTML>
      <H1>Starting Page for my_app</>
      <A href=<uri subapp/>>Start subapp</A>
    </HTML>
  </defmethod>
</defclass>
```

## HTML Form Inputs

User input comes from HTML form elements. All the basic HTML input controls are shown in the following example:

```
<defclass test_app>
  <defmethod start>
    <FORM action="/inspect_inputs?">
      <H3>Basic Input</>
      <INPUT type="text" name="basic_input" value="fred"/>

      <H3>Long Desc</>
      <TEXTAREA name="long_desc">some long
text with a linebreak</TEXTAREA>

      <H3>Hidden Field</H3>
      <INPUT type="hidden" name="hidden_field" value="hidden value"/>

      <H3>Secret</H3>
      <INPUT type="password" name="secret" value="xxyyzz"/>

      <H3>Planets</H3>
```

```
<INPUT type="checkbox" name="planets" value="mars" checked=1/>Mars
<INPUT type="checkbox" name="planets" value="jupiter" />Jupiter

<H3>Fruit</H3>
<INPUT type="radio" name="fruit" value="apple" checked=1/>apple<br/>
<INPUT type="radio" name="fruit" value="pear" />pear<br/>

<H3>Tool</H3>
<SELECT name="tool">
  <OPTION value="wrench">wrench</option>
  <OPTION value="hammer" selected=true>hammer</option>
  <OPTION value="screwdriver">screwdriver</option>
</SELECT>

<H3>Tools_1</H3>
<SELECT name="tools_1" multiple=true>
  <OPTION value="wrench" selected=true>wrench</option>
  <OPTION value="hammer" selected=true>hammer</option>
  <OPTION value="screwdriver">screwdriver</option>
</SELECT>

<H3>Button</H3>
<INPUT type="submit" name="button" value="Do Submit"/> <BR/>
<INPUT type="submit" name="button" value="Another Submit"/>

</FORM>
</defmethod>

<defmethod inspect_inputs all_inputs='other_keyed_args'>
  all_inputs
</defmethod>
</defclass>

Output as ConciseXML:
<thing
  basic_input="fred"
  long_desc="some long\ntext"
  hidden_field="hidden value"
  secret="xyyyzz"
  planets="mars"
  fruit="apple"
  tool="hammer"
  tools_1=<vector "hammer" "wrench"/>
  button="Do Submit"
/>
```



Chapter 33 on Water Bridge to JavaScript describes how to add JavaScript to HTML pages.

Water App uses the standard HTML form elements, so any book on HTML serves as a good reference for creating Water App forms.

## An Example Application

The following example defines a new Water App called `rotating_ad_display`. The start page displays a changing ad. The ad is displayed and rotated through the `display_next` method of the `ad_rotator` object. Every time the application is displayed, a new ad is shown.

```
<defclass ad_rotator ads flag_count=0 >
  <defmethod display_next>
    .set flag_count=
      .flag_count.<plus 1/>.<remainder .ads.<length/> />
  />
  <H1><do .ads.<get .flag_count/> /></H1>
</defmethod>
</defclass>

<defclass rotating_ad_display
  my_ads=
    <ad_rotator <vector "Drink Water" "Eat Well" "Get good sleep" /> />
>
  <defmethod start>
    <HTML>
    <do .my_ads.<display_next/> />
    </HTML>
  </defmethod>
</defclass>
```

`rotating_ad_display` has a field named `my_ads`, which is an instance of an `ad_rotator` that is shared across all users of this application. The `display_next` button is called when the start page is displayed, and, therefore, the next ad in the ad list is shown on every request.

Any other feature from the other Water solutions can be used in conjunction with Water App. Water App represents one of the most flexible ways to build Web-based applications.

## Summary

When building a Water App, you can use all the other Water solutions. Water App is really just a design pattern for building Web-based applications. No new concepts were introduced except for the convention of calling the `start` method when an application is shown. The same core building blocks of `defclass` and `defmethod` are used here a novel way to create flexible Web-based applications.