

Chapter 1

Introduction to Pure Web Service Programming

IN THIS CHAPTER

- ◆ Understanding why Water was created
- ◆ Creating your first Water program
- ◆ Creating a simple Water application
- ◆ Creating a Water application server
- ◆ Creating and calling a Web service

XML IS A SYNTAX FOR REPRESENTING DATA. XML is currently used for many different purposes: data representation, function calling, data typing, data storage, and many other uses. XML is being used on the client, the middle-tier, and back-end legacy systems. XML parsers are widely available for most programming languages, so why do we need a new language to work with XML?

Why Do We Need Another Language?

Before answering this question, consider the following: When you code in your favorite language, do you think about the language parser that parses your source code? In most cases, you shouldn't need to because the parser is transparent to the programmer. Most importantly, the parser does not have to be called explicitly. The compiler or interpreter calls the parser for the user. The designers of Water believe that working with XML should be as smooth and transparent as working with the native data and functions of a language. This isn't possible unless you use a programming language that is native to XML. If XML is used to represent general-purpose data, then a general-purpose language is required to manipulate this data. Water fills this role and is specifically designed for XML and Web services. Water is pure Web service programming language because Water uses a ConciseXML syntax that is both compatible and a superset of XML 1.0.

A programming language that is pure to Web services and XML has significant implications. The average size and complexity of the Web services and applications built using Water are usually between 10 and 100 times smaller than the same applications written in other leading languages. We have seen that even new Water developers can find tremendous leverage in using Water. Although Water can simplify the HTML presentation layer, Water excels at building secure, complex, distributed applications and services. Water also has many features that appeal to those developing expert systems and artificial intelligence systems.

Building Web services and applications on either a J2EE or .NET platform is very complex because a dozen or more languages and technologies are required. A J2EE application, for example, often uses the following languages: XML, HTML, SQL, XSLT, JSP, Java servlets, JavaBeans, JavaScript, JavaDoc, CSS, and shell scripts. Most of these technologies and languages are very different from one another. Each one has a different syntax, a different object model, a different way to represent data, and a different way to debug programs. These differences add significant complexity to the software development process. Complexity is the biggest problem facing software developers today; it increases cost, lowers performance, affects security, and reduces reliability. Water is a pure XML language designed to reduce dramatically the complexity of building Web services and applications. Water can be used in place of all the languages and technologies discussed earlier. Water is a single unified tool that embodies the “Learn once, use everywhere” philosophy.

Water’s ease of use means that middle-school students as well as college students can learn it as their first programming language. Experienced developers, particularly those familiar with dynamic languages, will find Water very easy to learn.

What is Water’s Paradigm?

If Water is a general-purpose programming language, what programming paradigm does it follow? Object-oriented? Functional? Declarative? Data-centric? Procedural? Dynamic?

Although Water might be considered most similar to an object-oriented (OO) paradigm because everything is an object, Water breaks away from the traditional rigidity of OO, providing a much more flexible model. Water actually supports all of the above disciplines. It’s possible to write a Water program that avoids using inheritance, if you so choose. Water also blurs the distinction between declarative and procedural languages.

Many languages influenced the design of Water, including Lisp, Java, JavaScript, Self, Scheme, SmallTalk, ML, HTML, and XML.

Water Development Tools

The Steam IDE from Clear Methods is the first commercial Integrated Development Environment (IDE) for the Water language. A free version of the Steam IDE is available for download at www.waterland.org. The initial screen is shown in Figure 1-1.



The installation details are described in Chapter 36.



Figure 1-1: An Integrated Development Environment (IDE) for Water from Clear Methods

This IDE may be used for running all the code examples in this book. The IDE contains three major panes: Source Code, Object Inspector, and the HTML browser. Water code is typed into the Source Code pane. To execute the code, click the green Execute button. The output is shown in the two panes to the right of the Source Code pane. The upper-right pane shows the output in text form. Clicking anywhere within the pane opens an Object Inspector pane showing the structure of the returned value. The lower-right pane is a primitive HTML viewer that shows the output value as rendered in HTML. The HTML viewer does not support user events such as clicking links, but clicking anywhere in the pane will show the HTML string.

Hello World

The first example is the obligatory “Hello World” shown in the following example. The first line is the Water source code and the output is shown to the right of the arrow.

```
"Hello, World!"  
→ "Hello, World!"
```

As you can see, the source code for the Hello World program is the same as the output from the program. In Water, executing a string returns a string. When executing a sequence of Water expressions, Water returns the value of the last one by default. Because there is only one here, the value of this one expression is returned, which is the string. I believe Water has the shortest “Hello World” program of any language.

Water includes a standard hypertext library that defines all the HTML tags. Although the following example just looks like HTML, a lot more is happening behind the scenes.

```
<H1>Welcome to Water!</H1>  
→ <H1>Welcome to Water!</H1>
```

The first line is a call to the constructor of the `H1` class object. This call creates an instance of `H1`. When the instance is converted to HTML for output, it appears the same as the original source code.

Notice that `H1` is capitalized. By convention, HTML calls are uppercase, while all other calls are lowercase.

Water Web Services

Although trivial examples are interesting, they make it difficult to show what a language can do. Water is not only simple, but also extremely flexible and powerful. Some of the most complex systems I have ever designed and built has been done in Water. For the remainder of the chapter, I will show you some of the very cool things that you can do in Water. If you get lost or confused, don't worry – the topics are covered in detail throughout the book.

For the remainder of this chapter, you see how to do the following by using Water:

- ◆ Create a two page HTML form based application
- ◆ Define a Web service interface
- ◆ Create a new Web service server
- ◆ Define a custom data definition in XML
- ◆ Create an XML object
- ◆ Implement business logic in Water
- ◆ Call a remote Web service over HTTP

These tasks will be accomplished using Water and without any graphical tools or code generation. If you think that's amazing, just keep reading the rest of the book. Water's elegance and power will impress even the most jaded cynic.

Creating a form

The first task is to build a simple two-page HTML application. The first page is an HTML form that asks the user to type in a last name and first name. The second page displays a message that says the person has been registered.

If you are familiar with HTML, this looks like a standard HTML form. The form's action says where to go next. The form has two INPUT fields named "fname" and "lname" with a default value. There is also a submit button labeled Go.

```
<FORM action="/results">
  First: <INPUT name="fname" value="Mike"/> <BR/>
  Last: <INPUT name="lname" value="Plusch"/>
  <INPUT type="submit" value="Go"/>
</FORM>
```



Chapter 32 on Water App describes all the type of form input.

Defining a Web service interface

The preceding example is also standard Water code using Water's hypertext library. Now let's create an interface to a Web service using Water Contract.

```
<defmethod registration fname lname/>
```



Chapter 5 on Water Contract describes defining interfaces using defmethod.

This interface is named `registration` and it takes two arguments named `fname` and `lname`. The arguments are required and can take any object. The next example shows the same interface, but both arguments are now optional and they each have a default value. Both argument values must be of type `string`.

```
<defmethod registration fname="Mike"=string lname="Plusch"=string/>
```

Now you'll create a complete Web service that combines the interface specification with an implementation for the Web service that displays an HTML form.

```
<defmethod registration fname="Mike"=string lname="Plusch"=string>
  <FORM action="/result">
    First: <INPUT name="fname" value=fname/> <BR/>
    Last: <INPUT name="lname" value=lname/>
```

```
<INPUT type="submit" value="Go"/>
</FORM>
</defmethod>
```



Chapter 19 on Water Method describes how to implement methods.

The content area of the `defmethod` call is the instance of `FORM`. Note that the input parameters `fname` and `lname` are the values for the `INPUT` fields.

The Web service can now be called as a local Water method.

```
<registration/>
```

Creating a new Web service server

To call the Web service through a standard browser, you need to install a local Water Web server on port 80. The following code creates a new HTTP listener on port 80:

```
<server thing/>
```

Now go to your Web browser and type the following address to display the registration form:

```
http://localhost/registration?
```



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

Now you'll need to create another Web service named `result` that takes a first name and last name and displays a confirmation page.

```
<defmethod result fname lname>
  <HTML>
    <do fname/> <do lname/> has been registered.
  </HTML>
</defmethod>
```



Chapter 17 on Water Flow describes `do`.

The `do` calls execute the Water code inside them. In this case, there is only a Water variable, so the `do` calls return the value of that variable. Clicking the Go button on the registration form now calls the `result` method and displays the page.

Creating an XML object and custom data definition

Water Object uses XML for creating instances. Here's an example of a simple two-dimensional coordinate:

```
<point x=5 y=10/>
```



Chapter 5 on Water Contract describes `defclass`.

Water Contract can be used to define a data definition for objects. The following example uses `defclass` for creating a class object `point` that has two fields, `x` and `y`:

```
<defclass point x y/>
```

The following example is a list of two coordinates using the `vector` object:

```
<vector>  
  <point x=5 y=10/>  
  <point x=3 y=3/>  
</vector>
```



Chapter 3 on XML object describes `vector`.

You can create a method that returns the following object:

```
<defmethod get_point>
  <vector>
    <point x=5 y=10/>
    <point x=3 y=3/>
  </vector>
</defmethod>
```

If you have a Water HTTP server running, you can type the following into the browser's address bar to run the Web service (see Figure 1-2):

```
http://localhost/get_point?
```



Figure 1-2: Browser screen shot showing XML being returned

Implementing business logic in Water

The following example adds a parameter named “number” to the Web service interface. It specifies that the `number` argument is optional when calling the service, and the value defaults to `false`. The parameter type is `<one_of integer false/>`, which means that the argument's value must be either an integer number or `false`.

```
<set positive_integer=
  <type.range_of min=0 number_type=integer/>
/>
<defmethod get_point a_number=false=<one_of positive_integer false/> >
  <set point_list=
    <vector>
      <point x=5 y=10/>
      <point x=3 y=3/>
    </vector>
  </set>
  <if> a_number.<and a_number.<less point_list.<length/> /> />
```

```
    point_list.<get a_number/>
  else
    point_list
</if>
</defmethod>
```

The method has two top expressions—a call to `set` and a call to `if`. The call to `set` assigns the variable named `point_list` to a vector holding two instances of `point`. The `if` expression tests `a_number`. If `a_number` is not `false`, and the number is less than the number of values in `point_list`, then the point at that position in the vector is returned. If `a_number` is not given, then the entire list is returned.



Chapter 17 on Water Flow describes `if`.

Chapter 4 on Water Type describes `one_of`.

Chapter 18 on Water Logic describes `and` and `less`.

Chapter 19 on Water Methods describes `length`.

The `get_point` could be called through the browser using a URL such as either of the following:

```
http://localhost/get_point?
http://localhost/get_point?a_number=1
```

Calling a Web service over HTTP

If you want to call the Web service from Water, create a `web` resource with the URI (URL) of the Web service, and assign it to a variable as in the following example.

```
<set point_1=
  www.<web "http://localhost/get_point?a_number=1"/>
/>
```



Chapter 12 on Water Web describes `www` and `web`.

The following example retrieves the result by executing the `point_1` symbol. The result is a string of ConciseXML.



Chapter 2 on ConciseXML describes the syntax used by Water.

```
point_1  
→ "<point x=3 y=3/>"
```

To transform a string of XML into a Water object, simply call `execute`.

```
<set a_point="<point x=3 y=3/>".<execute/> />
```



Chapter 8 on Water Import describes `execute`.

Executing the symbol will return the object.

```
a_point  
→ <point x=3 y=3/>
```

As an object, you can easily retrieve fields of the object. The following example returns 3, the value of the `x` field.

```
a_point.x  
→ 3
```



Chapter 6 on Water Path describes dot notation and how to retrieve values.

This chapter presented many different concepts that are covered in detail in the remaining chapters.

Summary

At this point, I hope you can start to see some of Water's potential. In this single chapter, I covered an amazing amount of functionality. You should not expect to understand all the examples – the remaining chapters will describe each of the areas in more detail. The Water code has a simple, concise syntax that should have been quite readable – there were no strange symbols or characters. Also notice that the examples were done with source code, and no graphical development tools were required. Hopefully you feel that you could become productive in Water very easily – I have only been using Water for a little over two years. I have found Water to be both a simple and powerful tool. There is a lot of promise around XML and Web services, but I feel that Water represents one of the few technologies that can deliver on that promise.

