

## Chapter 43

# Building a Water App for Frequently Asked Questions

### IN THIS CHAPTER

- ◆ Creating an application
- ◆ Creating links between pages
- ◆ Calling a method from an href
- ◆ Reading and writing objects to a file
- ◆ Storing instances in a class

THIS CHAPTER PULLS TOGETHER MANY OF THE LESSONS from previous chapters and describes how to build a Water application to create a simple Frequently Asked Questions (FAQ) list. The previous chapters explained topics using small, focused examples. This chapter presents a larger example that shows how everything combines together.

## Reviewing the Requirements

This application is a content entry application for Frequently Asked Questions (FAQ) data. The first page lists all the FAQ entries and two links. One link lets you add another FAQ entry; the other link saves the FAQ data to disk. The FAQ entry form shows input fields for the question and answer as well as an Add FAQ button that adds the FAQ and shows another FAQ entry page. The FAQ entry page also shows the last FAQ for reference, as well as a link to go to the first page. The content must be stored on disk in an XML format. Figure 43-1 and Figure 43-2 show the pages of this program.

To run the application, first copy and paste the full program into your IDE and press Execute. Second, open a Web browser and go to address `http://localhost/listing?`.



Figure 43-1: First page in the FAQ application

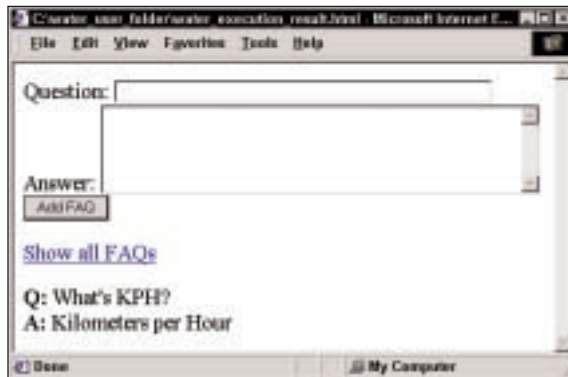


Figure 43-2: Adding a question and answer to the FAQ list

## Implementation

The following listing shows the complete code for this application. A single object named `faq` represents the entire application and contains all the methods and data for this application. The rest of the chapter describes the following Water program.

```
<defclass faq question="" answer="">

<defmethod listing alert_msg=null>
  <hypertext>
    <H2>Frequently Asked Questions (FAQ's)</>
    <P>Total of <do .<length/> /> entries.</P>
    <do.<for_each returns='all'> value </> />
```

```
<P><A href=<uri <edit/> /> >Add FAQ</></>
<P><A href=<uri <save/> /> >Save to disk</></>
  <do alert_msg/>
</hypertext>
</defmethod>

<defmethod details>
  <hypertext>
    <DIV><B>Q:</B> <do .question/> </DIV>
    <DIV><B>A:</B> <do .answer/> </DIV>
  </hypertext>
</defmethod>

<defmethod edit>
  <FORM action=<uri <add/> /> >
    Question: <input name="question" size=50/><BR/>
    Answer: <textarea name="answer" cols=50 rows=5/>
    <input type="submit" value="Add FAQ"/>
    <P><A href=<uri <listing/> /> >Show all FAQs</A></P>
    <do .<last/> />
  </FORM>
</defmethod>

<defmethod add question answer>
  <faq question answer/>
  faq.<edit/>
</defmethod>

<defmethod save>
  <set the_faqs=""/>
  faq.<for_each>
    <set the_faqs=the_faqs.<concat value.<to_concise_xml/> /> />
  </for_each>
  faq_file.<set content=the_faqs/>
  faq.<listing
    alert_msg=<P><do faq.<length/> /> records saved in <do faq_file/></P>
  />
</defmethod>

<defmethod init>
  ._parent.<insert _new_object/>
  _new_object
</defmethod>

<defmethod to_html>
```

```
.<details/>.<to_html/>
</defmethod>

_new_object.<set data_file=filesystem.<file "C:/faqs.h2o"/> />

<if> .data_file.<exists/>
  .data_file.<execute/>
else
  <do>
    <faq "What's MPH?" "Miles per Hour" />
    <faq "What's KPH?" "Kilometers per Hour" />
  </do>
</if>

<server faq/>

</defclass>
```

## Walking through the Design and Implementation

The rest of the chapter describes each part of the program. The first line is a `defclass` that creates a class object named `faq`. Most applications have a single top-level object that contains all the methods and data for the program. The instances of `faq` should contain fields for a question and an answer.

```
<defclass faq question="" answer="" />
```

Notice that the preceding line is a complete Water call that ends with `/>`. In the program, though, the first line is left open so that all the methods are defined within the content area of the `defclass`.



Chapter 32 on Water App describes building a Water application using `defclass`.

---

The following example shows the outline for the `faq` class by describing the interface for the methods, not the implementation. All the methods are defined in the content area of the call to `defclass`.

```

<defclass faq question="" answer="">
  <defmethod listing alert_msg=null/>
  <defmethod details/>
  <defmethod add question answer/>
  <defmethod edit/>
  <defmethod save/>
  <defmethod init/>
  <defmethod to_html/>
</defclass>

```

The `listing` method takes an optional `alert_msg`. It returns a hypertext object with a title, the number of entries, a list of every entry, and two links.

```

<defmethod listing alert_msg=null>
  <hypertext>
    <H2>Frequently Asked Questions (FAQ's)</>
    <P>Total of <do .<length/> /> entries.</P>
    <do .<for_each returns='all'> value </> />
    <P><A href=<uri <edit/> /> >Add FAQ</></>
    <P><A href=<uri <save/> />>Save to disk</></>
    <do alert_msg/>
  </hypertext>
</defmethod>

```

The next line shows a link that calls the method `create`. A question mark means that the object should be called.

```

<A href=<uri <edit/> /> >Add FAQ</>
→ <A href="/edit?">Add FAQ</>

```

The method `details` formats a particular instance of `faq`. The `do` calls `insert.question` and `insert.answer` into the HTML. `insert.question` is equivalent to `insert._subject.question`.

```

<defmethod details>
  <hypertext>
    <DIV><B>Q:</B> <do .question/> </DIV>
    <DIV><B>A:</B> <do .answer/> </DIV>
  </hypertext>
</defmethod>

```

The `create` method returns an HTML form with an action `"/add"` and two input fields named `question` and `answer`.

```

<defmethod edit>
  <FORM action=<uri <add/> /> >
    Question: <input name="question" size=50/><BR/>
    Answer: <textarea name="answer" cols=50 rows=5/>
    <input type="submit" value="Add FAQ"/>
    <P><A href=<uri <listing/> /> >Show all FAQs</A></P>
    <do .<last/> />
  </FORM>
</defmethod>

```

The last line of the FORM is `<do .<last/>/>`. The call returns the last value of the vector and automatically formats it as HTML.

The `add` method acts like a controller for the application. The method gets called when the user presses the Add FAQ button. That button has `type="submit"`, which means that clicking it will request the value of the form action, `<add/>`, from the server.

```

<defmethod add question answer>
  <faq question answer/>
  .<edit/>
</defmethod>

```

The first line of `add` creates a new instance of `faq` using the form's value for the question and answer. The second line says to call `edit`, which redisplay an entry form.

The `save` method creates an XML string for all the FAQs and saves the string to a file. The last expression calls the `listing` method with an `alert_msg` that says how many records were saved.

```

<defmethod save>
  <set the_faqs=""/>
  faq.<for_each>
    <set the_faqs=the_faqs.<concat value.<to_concise_xml/> /> />
  </for_each>
  data_file.<set content=the_faqs/>

  faq.<listing
    alert_msg=<P><do faq.<length/> /> records saved in <do faq_file/></P>
  />
</defmethod>

```

Every class object (or actually any object) can create a custom constructor method that says what should happen when a new instance is created. The constructor method is named `init`, short for instance initialization.

```
<defmethod init>
  ._parent.<insert _new_object/>
  _new_object
</defmethod>
```

The custom `init` method in the preceding code adds the new instance as a vector field on `faq`. The first line is equivalent to `faq.<insert _subject/>`. Most `init` methods will have a final line with `_subject`, because most `init` methods will return the new instance.

The `to_html` method says that to format the `faq` object, call the `details` method. That method returns a hypertext object that then needs to be converted to text using `to_html`.

```
<defmethod to_html>
  .<details/>.<to_html/>
</defmethod>
```

The following line sets a class field named `data_file` to the file that holds the FAQ data. `data_file` is used by the `save` method. As with all other expressions in the content area of `faq`, the expression gets executed only once when `faq` is defined.

```
_new_object.<set data_file=filesystem.<file "C:/faqs.h2o"/> />
```

The `if` expression in the following code either executes the contents of the `data_file` or creates two new instances:

```
<if> .data_file.<exists/>
  .data_file.<execute/>
else
  <do>
    <faq "What's MPH?" "Miles per Hour" />
    <faq "What's KPH?" "Kilometers per Hour" />
  </do>
</if>
```

The `faq` program starts a custom HTTP server to serve the application. Starting up a server might instead be put into a separate configuration file.

```
<server faq/>
```

The entire application was just a single `Water` object, and each page was a field in the object. At this point, you should be able to extend the program or start writing your own `Water` program.

## Summary

This chapter described how to build an application for entering new data and persisting it to a file. We used many of the concepts from throughout the book. Hopefully you have been trying out some examples as you have progressed through the book, and have been getting comfortable with the syntax and development environment.

At this point you should try to build some small applications to reinforce the concepts that you've learned. Water represents a fundamental simplification in Web programming.

If you have been using other languages, you will likely find yourself having to *unlearn* things. In many cases, this is harder than just learning it from scratch. If you are learning Water for the first time, you won't really appreciate how much simpler it is to program in Water. With some luck, perhaps you will never have to know.