

Chapter 11

Water Filesystem: Using Files and Folders

IN THIS CHAPTER

- ◆ Reading and Writing a File
- ◆ Importing and Exporting from a File
- ◆ Calling a File as a Method
- ◆ Using Folders
- ◆ Processing a Batch of Files
- ◆ Importing Water source code files

A FILESYSTEM IS A CONVENIENT PLACE for storing and managing persistent data such as code, content, and data. Water Filesystem provides a simple, clean, cross-platform interface to the local filesystem. Water Filesystem has classes and methods for working with files and folders (directories). The files and folders are filesystem resources that can only be created from another filesystem resource. This follows the Water Access Rights (Chapter 29) security policies, which constrain access to the filesystem. Every file containing Water code can be called like a method.

Reading from a File

```
filesystem.<file <uri protocol="file" path="C:/temp/foo.h2o"/> />
```

The preceding example uses two new objects: `filesystem` and `file`. The `filesystem` object is the single resource that represents the authority for accessing the filesystem. The `file` class requires a subject that is a resource, and a first argument that is an identifier. Calling the `file` class will return an instance of `file` that is a new file resource. This resource can be used to manipulate the file.

The path to a file is also an accepted identifier for creating the `file` resource. The following example represents the same file resource as the preceding example.

```
filesystem.<file "C:/temp/foo.h2o"/>
```

The resource can be set to a local variable and then used to retrieve the contents of a file as a string.

```
<set F=filesystem.<file "C:/temp/foo.h2o"/> />  
F.content
```

Creating a file resource does not verify that the file exists on disk. That can be done by calling the `exists` method on a file resource.

```
filesystem.<file "C:/temp/foo.txt"/>.<exists/>
```

A few other fields provide information on a file.

```
F.last_modified_on → <datetime 2002 10 10 7 30/>  
F.last_modified_by → "mplusch"  
F.created_on → <datetime 2002 5 10 10 30/>
```

Writing to a File

To set the contents of a file, call `set` with a `content` argument that has the new value for the file.

```
F.<set content="new content"/>
```

Now the content of the `F` file will be the string "new content".

```
F.content → "new content"
```

Any type of object can be given as the content. Any non-string value will be converted to a string by calling the `to_xml` method.

```
F.<set content=5/>  
F.content → "<do>5</do>"
```

Calling `to_xml` on any hypertext object will call the `to_html` method on the object.

```
F.<set content=<html><body>an HTMLpage</></> />  
F.content → "<html><body>an HTMLpage</body></html>"
```

To insert an object into a file, call the `insert` method that takes the value to insert.

```
F.<insert "new content"/>
```

By default, `insert` puts the new content at the end of the file. `insert` also takes an optional `at_key` argument that has the character position of where to insert the new string. The following example first sets the content of the file to `<thing color="red"/>`, and then inserts the string `<?xml version="1.0"?>` at the start of the file.

```
F.<set content=<thing color="red"/> />
F.<insert at_key=0
    <?xml version=1.0 ?>
/>
```

The file will now have the following two lines:

```
<?xml version="1.0"?>
<thing color="red"/>
```

To get the file's identifier, which is the URI/URL of the file, get the `a_uri` field from the file resource.

```
<file "C:/temp/foo.h2o"/>.a_uri →
<uri protocol="file"
    path=<vector "C:" "temp" "foo.h2o"/>
/>
```

Including Files

Water Filesystem can directly execute the contents of a file calling the `execute` method. This is the way to include a source code file in a Water program or load a data set into a Water object.

```
filesystem.<file "C:/temp/foo.h2o"/>.execute/>
```

The `execute` method on a file gets the content of a file as a string and executes it.



Executing in the current environment: `execute` takes an optional first argument that is the environment to execute the content.

```
a_file.<execute _environment/>
a_file.content.<execute _environment/>
```

Importing and Exporting a File

Water Filesystem can be combined with Water Export to save a Water object in a CSV file.

```
<set F=filesystem.<file "C:/temp/data.csv"/>/>
F.<set content=<vector <vector "pants" "red" "XL"/>
      <vector "shirt" "blue" "M"/>
      />.<to_csv/>
  />
F.content
```

Output as text:

```
pants,red,XL
shirt,blue,M
```

Water Filesystem can be combined with Water Import to convert a CSV file into a Water object.

```
F.content.<csv_to_objects/>
```

Output as ConciseXML:

```
<vector
  <vector "pants" "red" "XL"/>
  <vector "shirt" "blue" "M"/>
/>
```

Calling a File as a Method

Water Filesystem can treat a file containing Water code as a callable method. The content of the file can be executed to return an object. The value of the last top-level expression in the file is returned. First, create a new file with the contents of "`<defmethod foo x y > x.<plus y/> </>`".

```
filesystem.<file "C:/foo.h2o"/>.
  <set content="<defmethod foo x y > x.<plus y/> </>" />
```

Now the file resource can be executed to define a method.

```
filesystem.<file "C:/foo.h2o"/>.<execute/>
<foo 5 10/> → 15
```

The file can contain more than one top-level expression, but only the value of the last expression is returned when `execute` is called.

Creating Folders

Water Filesystem has both file and folder resources. Creating a folder resource is similar to creating a file resource. The identity of a folder is given by either a uni

or a string of the folder path. Each line in the following example represents the same folder resource. The creation of a `folder` requires a subject of the `filesystem` resource or another `folder` resource.

```
filesystem.<folder "C:/temp"/>
filesystem.<folder "C:/temp/foo.txt"/>
filesystem.<folder <uri> C:/temp/ </uri> />
filesystem.<folder <uri path="C:/temp/" />/>
filesystem.<folder "C:"/>.<folder"temp"/>
```

To list the content of a folder, ask for the `content` field of a `folder` resource. It returns a vector of the folder and file resources in the folder's content.

```
a_folder.content
```

Output as ConciseXML:

```
<vector>
  a_folder.<file "foo.h2o"/>
  a_folder.<folder "bar"/>
</vector>
```

In the preceding example, `a_folder` contains one file named `foo.h2o` and one folder named `bar`.

A file resource can be created by using a folder resource as the subject. This effectively returns a file within a folder.

```
a_folder.<file "foo.h2o"/>
```

A file resource can also be returned through its position in the `content` field. This works because the `content` field of a `folder` resource is an object with vector fields containing file or folder resources.

```
a_folder.content.0
```

A folder resource can be the subject to create another folder resource. A Water Path that represents a series of folder resources looks like a folder path. The following three lines represent the same file resource, but they get to the file in different ways.

```
filesystem.<file "C:/temp/doc/foo.txt"/>
filesystem.<folder "C:/temp/doc"/>.<file "foo.txt"/>
filesystem.<folder "C:/temp"/>.<folder "doc"/>.<file "foo.txt"/>
```

The `exists` method can be used to check if the identified folder exists on disk. It returns `true` or `false`.

```
<folder "C:/temp"/>.<exists/>
```

Security Access

The `filesystem` object represents the ability to do file operations on the local filesystem. It is a very powerful resource because it gives the holder of the resource the access and rights to the entire filesystem.

A particular program or user might not be given the rights to the `filesystem` resource. They might only be given access to a narrow piece of the local filesystem. The following example sets a variable named `temp_folder` to the `temp` folder.

```
<set temp_folder=filesystem.<folder "C:/temp"/> />
```

This variable represents the permission, or the access right, to the `temp` folder. It does not let the user access any other folder outside that folder.

```
temp_folder.<file "foo.h2o"/>
```

This makes a `folder` resource similar to sharing a particular folder in the filesystem. Another machine can only see that folder and any resources within that folder.

Chapter 29 on Water Security shows how a `view` object can restrict access within a particular folder to only certain fields and methods. These restrictions can implement any business rule. A user, for example, might be able to create a new file, but not see any files except the ones that they created within the last hour.

The `identifier` field will return a `uri` object that represents the location of the `folder` resource.

```
<folder "C:/temp"/>.a_uri →  
<uri path=<vector "C:" "temp"/> />  
<
```

See the Water Solution Reference on the www.waterlang.org site for the complete list of fields on a file resource.

Creating Files and Folders

To copy a file, call the `copy_file` method on the original file resource. A `file` resource for the new file is returned.

```
filesystem.<file "C:/temp.h2o"/>.<copy_file/> →  
filesystem.<file "C:/temp_copy.h2o"/>
```

Copy takes an optional argument that is the identifier for the new file. The identifier can be the string of the new file name, a `uri`, a `file resource`, or a `folder resource`. The following lines of code all create a copy of the `temp.h2o` file.

```
filesystem.<file "C:/temp.h2o"/>.<copy_file "temp2.h2o" />
filesystem.<file "C:/temp.h2o"/>.<copy_file <uri>C:/temp2.h2o</uri> />
filesystem.<file "C:/temp.h2o"/>.<copy_file filesystem.<file "C:/temp2.h2o"/> />
filesystem.<file "C:/temp.h2o"/>.<copy_file <folder "C:/temp"/> />
```

To rename a file, call `rename` on a `folder resource` and pass it the original file name and the new file name.

```
filesystem.<folder "C:/docs"/>.<rename "old_filename.h2o" "new_filename.h2o"/>
```

To create a new folder on disk, call the `create` method on a `folder resource`.

```
filesystem.<folder "C:/temp"/>.<create/>
```

The `create` method also works on a `file resource`—it creates an empty file.

```
filesystem.<file "C:/temp/foo.txt"/>.<create/>
```

Batch File Processing

If you have a list of files that require some processing, call `for_each` on the list of files. The content argument of `for_each` contains the expressions to execute for every file. The following example returns a vector where each value in the vector is the content of a file as a string.

```
<set content_from_files=<vector/> />
filesystem.<folder "C:/temp"/>.<content.>
  <for_each>
    <if> value.<is_a file/>
      content_from_files.<insert value.content/>
    </if>
  </for_each>
content_from_files
```

In the previous example, `for_each` iterates over the content of a folder. For each file or folder in the directory, insert the content of a file at the end of vector

`content_from_files`. In the content of a `for_each` call, the variable `value` holds the current object that is retrieved in the subject.

The following example follows the same basic pattern as the last example. This one converts any HTML file into an XHTML file. The same pattern could be used for any kind of file conversion.

```
filesystem.<folder "C:/temp"/>.content.  
<for_each>  
  <if> value.<is_a file/>.<and value.<extension/>.<is "html"/> />  
    value.<set content=value.content.<html_to_xhtml/> />  
  </if>  
</for_each>
```

The example tests to see if the value is a file and if the file extension is `html`. If so, then the new content of the file is set to the original content of the file converted to XHTML. XHTML is a newer version of HTML that conforms to XML 1.0 syntax. Water comes standard with an `html_to_xhtml` method that converts any HTML file into a valid XHTML file. An example using that method is shown below.

```
"<BR> <SPAN> hello".<html_to_xhtml/> →  
"<BR/> <SPAN> hello</SPAN>"
```

Notice that the method closed up the HTML tags.

Processing multiple files is quite useful and Water's `for_each` method avoids the need to use tools outside of Water to handle file processing.

Summary

Water Filesystem presents a simple and elegant interface to the local filesystem. This makes it possible to easily store objects in the filesystem in any number of formats including XML, Water, and HTML. This chapter described how to read and write files using the `content` argument, and also showed how files could be used for import, export, and execution. Water Filesystem makes your local filesystem appear as simple Water objects.