

Chapter 10

Water Identifier and URI: Naming Resources

IN THIS CHAPTER

- ◆ Relating URIs and Water Identifiers
- ◆ Creating Identifiers from a Water expression
- ◆ Creating Relative Identifiers
- ◆ Remote and Local Identifiers

THE WEB IS A SET OF RESOURCES. Each Web site is a resource as well as a collection of related resources. Each Web resource has an identifier that names the resource. The standard identifier for Web resource is a Uniform Resource Identifier (URI, previously known as URL). A URI is a string that encodes information about a resource. A Water Identifier defines a `uri` class that uses an object to encode the information in a URI string. Treating a `uri` as an object makes it possible to manipulate and combine URIs in interesting ways.

Creating Identifiers

The following is a string object that represents a valid URI:

```
"http://www.waterlang.org"
```



An *identifier* is a name for a resource. An identifier is just data, and does not represent the permissions to use a resource.

Content Argument of uri is Executed as Hypertext

The content argument of a call to `uri` is treated as hypertext. This means that any text that is not within a call is treated as a string. Any calls in the content are executed and return a value.

```
<set F="www" />
<uri> http://<do F/>.waterlang.org </uri>
```

Output:

```
<uri> http://www.waterlang.org </uri>
```

The `do` call in the content argument above returns "www", the value of variable `F`. The string "www" is converted to HTML because the method `to_html` is called on the return value from any call inside a hypertext area.

```
"www".<to_html/> → www
```

You can find details on how to define and use hypertext arguments in Chapter 30 on Water Execute and Macros.



A *resource* represents the permission (rights and capabilities) to use that service. A resource can be created from a base resource and an identifier. A resource typically accesses some remote resource.

Identifiers can be created in a variety of ways by using the `uri` object. The first way to create a `uri` is passing a URI in the content argument when calling `uri`.

```
<uri> http://www.waterlang.org </uri>
```

A `uri` could also be created from a separate path of "C:/temp/foo.h2o" and a protocol of "file".

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

Using Water Identifier

An identifier can be used to create a resource. That resource can then be used to access some external resource. A `uri`, for example, can identify a file in the local filesystem. A `file` resource can be created if you have a file identifier and access to the filesystem.

```
filesystem.<file <uri>temp.h2o</uri> />
```

When the preceding line of code is executed, a file resource to the temp.h2o file is returned. You can then ask for the content of the file.

```
filesystem.<file <uri>temp.h2o</uri> />.content
```

There is a big difference between an identifier and a resource. The identifier is just a name and does not represent the actual live resource. A resource means that you have permission to make calls to that resource.



Chapters 11 and 12 on the Water Filesystem and Water Web describe resources for the filesystem and Web.

For Water programs, a uri is often found in the href attribute of a hyperlink.

```
<A href=<uri> http://www.waterlang.org </uri>>  
  The Water Language Site  
</A>
```

A uri is also found in the action attribute of a FORM object.

```
<FORM action=<uri>http://google.com</uri>.<uri search/> >  
  <INPUT name="query" value="search goes here"/>  
</FORM>
```

The uri in the action attribute will be translated into the following string:

```
"http://google.com/search"
```

More Ways to Create a URI

A Water Identifier can be created in multiple ways. In the following example, the uri is first created from a single string. The second line creates the same uri from the protocol and host arguments.

```
<uri> http://www.waterlang.org </uri>
```

Output as ConciseXML:

```
<uri protocol="http"host="www.waterlang.org"/>
```

A `uri` can be broken up into multiple parts. Each part is a field of a `uri` object. The first expression below is a `uri` created from a single string. The second expression is the same `uri` created with explicit fields.

```
<uri> http://user:info@waterlang.org:90/top_path/foo.html?bar=2&x=3#ref</uri>
```

Output as ConciseXML:

```
<uri protocol="http"
  user_info="user:info"
  host="waterlang.org"
  port=90
  path=<vector "" "top_path" "foo.html"/>
  query=<thing a="2" b="3"/>
  fragment="fragment"
/>
```

Assuming that the `uri` in the preceding example was set to the `U` variable, the following examples show how to retrieve parts of a `uri`.

```
U.protocol → "http"
```

```
U.user_info → "user:info"
```

```
U.host → "waterlang.org"
```

```
U.port → 90
```

```
U.path → <vector "" "top_path" "foo.html"/>
```

```
U.path_string → "top_path/foo.html"
```

```
U.<file_name/> → "foo"
```

```
U.extension → "html"
```

```
U.<file_name include_extension=true/> → "foo.html"
```

```
U.<is_folder/> → false
```

```
U.query_string → "bar=2&x=3"
```

```
U.query → <thing a="2" b="3"/>'
```

```
U.fragment → "fragment"
```

You can create a `uri` by using both a URI string and the individual pieces. The specific arguments override parts of the URI string. The following example shows how passing a protocol to `uri` will change the protocol of the original string.

```
<uri protocol="ftp"> http://www.waterlang.org</>
```

Output as ConciseXML:

```
<uri> ftp://www.waterlang.org </>
```

A `uri` can also be created by combining two other `uris`. The following example creates a new `uri` in `base_dev`. The second line creates another `uri`, which will take defaults from `base_dev`. This is extremely useful because there are multiple `uri` instances with the same base `uri`. If the base `uri` changes, then all instances created from that base will also change.

```
<set base_dev=<uri> http://localhost </uri>/>  
base_dev.<uri> languages/water </uri>
```

Output as ConciseXML:

```
<uri> http://localhost/languages/water </uri>
```

Creating Relative Identifiers

A URI has two major parts. The first part is the server identifier, which includes the protocol, host, and port. The second part is the relative identifier. This part includes the path, query, and fragment parts of the URI.

```
http://waterlang.org/product?language=english
```

In the preceding example, the server identifier is `http://waterlang.org` and the relative identifier is `product?language=english`. The first part represents where the resource is located and how to communicate with the resource. The second part represents the identifier within a particular server.



Relative Identifier: An identifier is relative if it lacks either the protocol or host fields.

The following is a relative `uri` because it is missing a protocol and host:

```
<uri> /product?language=english </>
```

The following is an absolute uri:

```
<uri host="waterlang.org"protocol="http"> /product?language=english </>
```

Creating a relative URI from a Water Expression

Creating a URI string can be quite complex. Water Identifier tries to simplify this process by creating a URI from a Water Path. In the following example, the Water Path `language.oo.Water` is translated into the equivalent URI string of `language/oo/water`.

```
<uri language.oo.Water /> → <uri>language/oo/Water </uri>
```

A relative Water Identifier is created from a Water Path. The first argument to `uri` takes a Water Path. The Water Path is converted to the equivalent URI string. For simple paths, the path separator is changed from a dot to a slash, but more complex paths are also supported.

```
<uri foo.<product prod_id=2 x=3 /> /> →  
<uri> foo/product?prod_id=2&x=3 </uri>
```

A Water call turns into a URI with a query string. Each keyword argument becomes a `name=value` pair in the query. If there are any arguments for a Water call, they get executed in the local environment and their value becomes the query value.

```
<set list=100/>  
<uri foo.<product price=list.<times 0.5/> />/>
```

Output as ConciseXML:

```
<uri> foo/product?price=50 </uri>
```

If a string URI and Water Path are both given, the Water Path will merge into the URI specified in the string.

```
<uri two.three > http://waterlang.org/one </uri>
```

Output as ConciseXML:

```
<uri> http://waterlang.org/one/two/three </uri>
```

A call in the Water Path will be translated into a query string.

```
<uri one.<two arg1="value1"/> >http://localhost?arg2=99 </uri>
```

Output as ConciseXML:

```
<uri> http://localhost/one?arg2=99&arg1=value1</uri>
```

A URI with a question mark indicates a URI with a query. A URI with a query is equivalent to a Water Path with a call. A URI without a query is equivalent to a Water Path that retrieves the value of a field.

When combining URIs, if the new `path_string` starts with a `"/`, the new path completely replaces the old path; otherwise, the new path is appended to the old path.

```
<uri> /one/two </uri>.<uri> /foo/bar</uri> →  
<uri> /foo/bar </uri>
```

The following example shows that the new path is appended onto the old path because the `path_string` does not start with a `"/`.

```
<uri> /one/two </uri>.<uri> foo/bar</uri> →  
<uri> /one/two/foo/bar </uri>
```

Local Water Identifier

Water Identifier supports both local and remote identifiers. Remote identifiers use the `uri` object. Local identifiers are used behind the scenes in Water. Every Water expression is an identifier. These expressions evaluate to a local resource, which is a local object. A local identifier can be explicitly created by calling `expr` with a content argument. The following example is a local identifier for `foo.bar`:

```
<expr>foo.bar</expr>
```

A Water expression can be executed by calling `execute` on it.

```
<expr>foo.bar</expr>.<execute/>
```

Summary

A URI is one of the key foundations to the World Wide Web. Water Identifier treats a URI as more than a simple string—it is a structured object with many fields. An instance of the `uri` class can be created from a string, individual parts, a Water expression, or a composition of multiple instances of `uri`. A `uri` includes all information required to make a remote request. Water Identifiers are used extensively in Water Filesystem (Chapter 11) and Water Web (Chapter 12).

