

## Chapter 16

# Water Registry: Listing Resources – UDDI

### IN THIS CHAPTER

- ◆ Publishing a Water Registry
- ◆ Accessing Resources in a Registry
- ◆ Understanding the similarities between a Registry and a Web site

**WATER REGISTRY IS THE WATER SOLUTION** for publishing a list of Web resources in a machine-readable format. A Water Server can publish the resources available on that server as well as other servers. The resource's URI/URL, type, and contract are available through Water Registry. This makes it possible for business partners to easily find and use resources.

UDDI and WS-Inspection are other standards providing this functionality. Water Registry provides compatible interfaces, but the Water Registry format is simpler and more flexible than either WS-Inspection or UDDI.

## Publishing a Water Registry

A Water Registry publishes a machine-readable list of resources where each resource has a type and a URI/URL that holds the location and protocol. A Water Registry can be on the same server as the resources listed in the registry, or a Water Registry might list resources located on another server.

Every Water object can be published as a Water Registry. Each field of an object is a resource. The following example shows four different types of resources. Each field holds a different type of resource:

- ◆ `closing_times` holds a vector of time objects
- ◆ `public_home_page` holds a remote resource representing the home page of `waterlang.org`
- ◆ `static_page` holds a hypertext object
- ◆ `get_product` holds a local Water method

```

<defclass waterlang
  closing_times=
    <vector <time 8/> <time 9/> <time 9/> <time 8/> />

  public_home_page=
    web.<web <uri>http://www.waterlang.org</uri> />

  static_page=<HTML>a static page</HTML>
>
<defmethod get_product product_id>
  <thing
    x101=<thing id="x101" name="Product A"/>
    x201=<thing id="x201" name="Product B"/>
  />.<get product_id/>
</defmethod>
</defclass>

```

Executing `waterlang.closing_times` returns a vector.

```
waterlang.closing_times → <vector <time 8/> <time 9/> <time 9/> <time 8/> />
```

The following example creates a new application server to serve the `waterlang` application. The `waterlang` application is the `root_object` of the server.

```
<server waterlang/>
```



Chapter 14 on Water Server describes how to create an application server.

---

The Water Registry for the root object is returned by requesting the following URI/URL:

```
http://localhost
```

I will now describe how to access the resources showing the registry list, you must access some of the fields of the server's root object.

## Accessing Registry Resources

Every resource in a Water Registry listing is accessible through a URI/URL. The following example shows the URI/URL to access the `waterlang.closing_times` object and the returned result.

```
http://localhost/closing_times →  
<vector <time 8/> <time 9/> <time 9/> <time 8/> />
```

The URI request returned the formatted string of the object. The other fields of `waterlang` are accessed in the same way.

```
http://localhost/public_home_page →  
<HTML> content of waterlang.org home page </HTML>
```

```
http://localhost/static_page →  
<HTML> a static page </HTML>
```

## Viewing a Method as a Resource

Accessing the `get_product` field that contains a method returns a string representation for a web resource with a `uri` and `contract`.

```
http://localhost/get_product →  
web.<web <uri>http://localhost/get_product</uri>  
    contract=<defmethod get_product product_id/>  
    />
```

## Converting a Resource into a Local Proxy Method

Executing this ConciseXML string returns a web resource object.

```
<set get_product_proxy=  
    "web.<web <uri>http://localhost/get_product</uri>  
        contract=<defmethod get_product product_id/>  
        />".<execute/>  
/>
```

The web resource can be called just like a normal method.

```
<get_product_proxy product_id="x101"/> →  
<thing id="x101" name="Product A"/>
```

The method could also be called using a URI/URL.

```
http://localhost/waterlang/get_product?product_id=101 →
<thing id=101 name="Product A"/>
```

## Using a Registry Listing

The following is the same class definition that was at the start of the chapter. The `defclass` and server creation is shown here for reference.

```
<defclass waterlang
  closing_times=
    <vector <time 8/> <time 9/> <time 9/> <time 8/> />

  public_home_page=
    web.<web <uri>http://www.waterlang.org</uri> />

  static_page=<HTML>a static page</HTML>
>
<defmethod get_product product_id>
  <thing
    x101=<thing id="x101" name="Product A"/>
    x201=<thing id="x201" name="Product B"/>
  />.<get product_id/>
</defmethod>
</defclass>

<server waterlang/>
```

The registry can be shown by accessing the following URI/URL:

```
http://localhost
```

**Output as ConciseXML:**

```
<defclass waterlang
  closing_times=web.<web "http://localhost/closing_times" type=vector />
  public_home_page=web.<web "http://www.waterlang.org" type=hypertext.HTML/>
  static_page=web.<web "http://localhost/static_page" type=hypertext.HTML/>
  get_product=web.<web "http://localhost/get_product" type=defmethod />
/>
```

The returned value is a Water Registry listing. The `waterlang` object is represented as a `defclass`, and each field of the object contains a `web` resource. The listing itself can be executed to return an object that is a local proxy for all the resources in the listing.

```
<set waterlang_proxy=  
    web.<web "http://localhost"/>.<execute/>  
</set>  
waterlang_proxy.closing_times  
<vector <time 8/> <time 9/> <time 9/> <time 8/> />
```

Water Registry is powerful because it dramatically simplifies the publication of machine-readable resources as well as automates the accessing of those resources.

## Understanding Water Registry

Every Web site is a human-browseable list of resources that users can browse. The user accesses resources by clicking a link. The hyperlink is a reference to a resource and the Web site enables the user to successfully navigate the site's resources.

A registry is a machine-browseable list of resources. When a machine or program visits a site, the program needs to understand the types of resources that are available to it.

A Water object can act like a registry because each field is a named container for an object, which is a resource. The resource might be a local object or a remote resource. A request to a resource is either calling the resource with arguments, or returning the content of the resource.

A Water Registry listing is a `defclass` where the value of each field is a resource. The XML representation of the resource could be executed by the client to create a remote resource. Requests can now be made on that resource.

If a field value contains a `web` resource, the XML representation is returned. That resource might list resources on another server. Because every resource listed in the registry has a type, the client can identify particular types of resources without having to request each one.

Retrieving a method resource, but not calling it, will return a Water Web resource. The resource contains a contract that describes the inputs and outputs when calling the resource.



**Water Type versus Water Contract.** A contract plays the role of a type. A method that is used to describe a contract must check to make sure the method has compatible inputs and outputs. Another way to view it is that a contract is a parameterized type and a method is its own type.

Methods for searching and filtering a Water Registry can be defined on any object. These methods are simply all the methods defined on the object.

Water Registry is the default format when a `defclass` or `folder` is returned by a server.

## Summary

This chapter described how to publish a Water Registry and access listed resources using concepts from almost every other chapter in *Water Web Services*. To date, the Web has been more human-friendly than machine-friendly. The simplicity of *Water Web Services* and *Water Registry* represents a significant step in making the Web machine-friendly.