

Chapter 3

Water Object: Representing Objects in XML

IN THIS CHAPTER

- ◆ Understanding Objects and Fields
- ◆ Defining Thing and Vector
- ◆ Using Primitive Objects
- ◆ Understanding Class and Instance
- ◆ Some Water Object Examples

WATER MAKES XML OBJECT-ORIENTED. This is an important step in the evolution of XML because this unifies two leading approaches to building software: Web Services and Object-Oriented Programming. Water Objects are used for data, logic, and presentation and every Water Object can be fully described in both XML 1.0 and ConciseXML.

Understanding Water Objects

Everything in Water is an object and objects form the core foundation for Water. Water has a very clean and flexible object model that can be used for many different purposes.

Object

A Water object is a collection of named parts. The word *object*, as used in this book, may be known by other words in other languages. The following words, for example, have similar meanings in other languages: record, instance, structure, data element, hash table, element, tuple, package, vector, array, collection, and set. Objects typically represent nouns. Some examples of business objects are as follows: a customer, an order, a receipt, a line item, a book, and so on.



An *object* has zero or more fields. Each field has a key and a value. The value can hold any object and the key can hold any object.

The parts of a Water object are called fields (see Figure 3-1). A field is known by many different names in other languages. For example, property, variable, column, part, slot, and relation are all terms from various languages that refer to the concept of a field. A book object might have fields of title, author, ISBN number, date published, and so on.

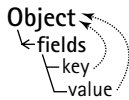


Figure 3-1: Diagram of an object

Field

Each field has a key and a value. The key of a field is a name for the field. You can think of a field as a named container. The container has a name, which is the key, and a value, which is the thing that the container holds. A field can only contain a single value, but that value can be any object. The same object can be the value of zero or more fields.



A *field* is a named part of an object that has a key and a value.

In Water, you always refer to XML text as either an XML object or an XML expression – never an XML document, in order to be more precise.

Thing Class

The following line creates a single object with two fields:

```
<thing x=10 y=20/>
```

XML Document versus XML Object

The XML standards define a file containing XML as an XML document. The precursor to XML was SGML, a document markup standard. This book refers to a file of XML, or any chunk of XML data, as an XML object. This is because the term document in common use refers to a particular type of structured data object that might have fields for author, last modified date, body text, and so on (for example, a word processing or spreadsheet document). The term XML object is used in place of XML document because an object is more generic than document, and object more precisely represents the broad application of XML data.

The preceding example is constructing a generic object. The object is an instance of class `thing`. The opening angle bracket of an XML 1.0 element indicates the creation of a new instance. `Water` refers to an element as a *call* because it represents either a method call or a constructor call.

In `Water`, `thing` is the most generic object. Creating an instance of `thing` has no fields by default, but any fields can be set within it.

This instance was created with two named fields: `x` and `y`. One field has a key of "x" and a value of 10. The other field has a key of "y" and a value of the integer 20.

Although this creates an instance of `thing`, the instance does not have a name and therefore cannot be referenced later.

Naming

To name an object, put it in some field. The key (or name) of the field will be the way you can refer to the object. You then refer to the object by name.

You could, for example, set an object to the value of a field using `set`.

```
<set some_thing=<thing x=10 y=20/> />
```

The preceding example names the instance of `thing` to `some_thing`. Now you can refer to `<thing x=10 y=20/>` by the name `some_thing`. Evaluating `some_thing` will return the instance.

```
some_thing → <thing x=10 y=20/>
```

Notice that assigning an instance to a name or referencing an instance by name does not create an instance—it simply refers to that instance and returns the instance when executed.



In the previous example, the keys of both fields were strings. When a key is shown on the left-hand side of the equal sign, it assumes the key is a string if the key starts with a character. You could have optionally described the keys as strings using double quotes: `<thing "x"=10 "y"=20/>`

The key of field does not have to be a string — it can be any object. One very common use of non-string keys is for vectors that have integer keys.

Vector

In Water, a `vector` holds a collection of objects. The objects in the vector may be of different types. Vectors are similar to arrays in other programming languages. In the following example, an instance of `vector` is created that contains three string objects.

```
<vector "steam" "water" "ice"/>
```



A vector is an object whose only fields have consecutive integer keys starting at 0. Vectors are useful for holding an ordered list of objects. Vectors can also be used to represent unordered lists.

The objects in a vector are stored in integer keys that start with 0. The vector instance below explicitly gives the integer keys for the fields.

```
<vector 0="steam" 1="water" 2="ice"/>
```

If you do not explicitly give the keys in a vector, the values are assigned to integer keys in the order in which they appear, starting with 0.

```
<vector "steam" "water" "ice"/>
```

Is equivalent to:

```
<vector 0="steam" 2="ice" 1="water"/>
```

Notice how the last line specifies the keys out-of-order. That is accepted in Water because all the keys are specified in the construction of the vector. The integer fields are completely given — there are no gaps in the integer keys and the first key is 0.

A purchase order, for example, might have a field named `line_items` that contains a vector of `line_item` objects.

```
<purchase_order
  po_number="1001"
  line_items=<vector
    <line_item line=1 item="soap" quantity=2/>
    <line_item line=2 item="ice" quantity=3/>
  />
/>
```

A vector can hold different types of objects. The following vector, for example, holds five objects: a string "water", an integer number 100, a decimal number 42.001, a boolean true, and null.

```
<vector "water" 100 42.001 true null/>
```

A vector that holds only vectors could be considered a two-dimensional vector.

```
<vector <vector "a" "b"/> <vector "c" "d"/> />
```

An empty vector does not contain any objects. This is an empty vector:

```
<vector/>
```

The following vector holds three instances of `thing`. Each instance has both an `x` and `y` field.

```
<vector
  <thing x=10 y=20/>
  <thing x=5 y=100/>
  <thing x=20 y=200/>
/>
```

In the preceding example, `<thing x=10 y=20/>` is stored in the field with key 0. The highest integer key is 2, which holds the value `<thing x=20 y=200/>`.

You can think of a vector as a constrained version of `thing` because a vector can only have integer keys starting at 0. If you need an object with any integer fields (objects that have integer fields but do not start at 0, or that are not consecutive), use `thing`; although a `thing` can act like a vector, it does not have the constraint on keys that a vector has. The following example uses non-consecutive integer keys.

```
<thing 5="ice" 2="water" 10="steam"/>
```

Because `thing` is more general than `vector`, you could use `thing` in place of a vector. For example:

```
<thing "ice" "water" "steam"/>
```

The preceding example has the same fields as the following example.

```
<vector "ice" "water" "steam"/>
```

Primitive Objects

Everything in Water is an object – even things like numbers and strings are objects. This is a simplification that makes it possible for all things to behave in a similar manner.



Primitive objects are objects that do not have any fields. There are three primitive objects (`true`, `false`, and `null`) as well as four classes that have primitive objects as instances (`number`, `boolean`, `character`, and `string`).

The following are examples of primitive objects:

```
<char 'w' />
100.0
10
true
false
"a string"
null
```

Nested Objects

Objects can be associated in any way to create many different data structures. This is possible because objects have fields, every field has a key and a value, and the value can contain any object.

Circular Data Structures

Objects can be nested to any level, and can refer to objects up the hierarchy. Therefore, Water objects can represent circular data structures (loops) that are fully supported. The following is an example of a circular data structure where an employee is its own manager.

```
<set employee_bob=<employee name="Bob" /> />
employee_bob.<set manager=employee_bob/>
```

The following is a `vector` that contains two non-primitive objects:

```
<vector <thing name="one"/> <thing name="two"/> />
```

Defining Class and Instance

In the previous example, I used two built-in Water classes: `thing` and `vector`. In most applications, you will create objects of a custom class.

To create an instance of a class, the class needs to be defined. Defining the class of a data structure is discussed in detail in Chapter 5, but a simple class definition can be made by using `defclass`. `defclass` is a Water method for defining a class. The name of the class is given as the first argument.

The line below, for example, defines a class object named `employee`.

```
<defclass employee/>
```



A *class* is a prototype for creating instances. A class is a role that an object can play, and any object can play the role of a class. The class holds data and methods that are shared by all the instances. By default, a class does not know its instances.



An *instance* is an object that was created from a class. An instance belongs to a class if the instance has a parent that is the class.

An instance is created by calling a class. You can create an instance of `employee` by calling the `employee` class.

```
<employee  
  fname="Mike"  
  lname="Plusch"  
  hired_on=<date 2000 10 5/>  
>
```

The preceding object is an instance of the `employee` class. The instance has three fields: `fname`, `lname`, and `hired_on`. The `hired_on` field has a value of a date instance `<date 2000 10 5/>`, which represents October 5, 2000. By default, an instance can have fields that are not defined in its class.

Water Object System

Water has a simple and powerful object system. The object system is a multi-role object system where a single object can play multiple roles. This chapter introduced the roles of class and instance. The Water object system has many features including multiple inheritance. The Water Object System is described in Chapter 22.

In Water, `hypertext` is a class, and every `hypertext` tag is an instance of `hypertext`. The following, for example, is an instance of class `INPUT`. It has one field named `value` that holds the string "Water".

```
<INPUT value="Water"/>
```

The content argument of most HTML objects has a special kind of execution called `'ek_hypertext'`. This makes it possible for Water to support both structured and unstructured data. If the content has an execution kind of `'ek_hypertext'`, continuous text outside of any elements is treated as a string, and any elements are executed normally.

```
<H1> this is one string in field 0
  <do "this element is executed. The value appears in field 1"/>
  this is another string in field 2
</H1>
```

Examples of Water Objects

The following example defines three classes and creates an instance of `book`:

```
<defclass book/>
<defclass author/>
<defclass chapter/>
<book
  title="Water Book"
  a_author=<author fname="Mike" lname="Plusch"/>
  published_on=<date 2002 11 5/>
  chapters=<vector
    <chapter name="Introduction"/>
    <chapter name="Objects"/>
    <chapter name="Methods"/>
  />
/>
```

The preceding object is an instance of the `book` class. The instance has four fields: `title`, `a_author`, `published_on`, and `chapters`. The `a_author` field has an instance of `author` in its value. The `chapters` field contains a vector. The vector contains a list of `chapter` instances.

The class definitions of `story`, `author`, and `chapter` could have specific fields defined, which I discuss in Chapter 5. After a simple class is defined, an instance of that class can be created with any number of named fields.

The following example represents an instance of `rectangle`:

```
<defclass rectangle/>
<defclass coord/>
<rectangle
  top_left=<coord x=10 y=50/>
  height=100
  width=200
  border=3
/>
```

The preceding code represents an instance of `rectangle` that has a `top-left` field containing a coordinate. The `height`, `width`, and `border` fields each contain an integer.

The following example defines a `grocery_list` class and creates an instance of `grocery_list` that contains an ordered list of the strings "apple", "milk", and "pears". The fields have integer keys of 0, 1, and 2.

```
<defclass grocery_list/>
<grocery_list
  0="apple"
  1="milk"
  2="pears"
/>
```

The integer keys were explicitly specified using `0="apple"` and `1="milk"`. If you want the values to be assigned to integer keys automatically, you can use the special `_promote` parameter in the `defclass` and give it the value `'other_unkeyed_args'`.

```
<defclass grocery_list _promote='other_unkeyed_args'/>
<grocery_list
  "apple"
  "milk"
  "pears"
/>
```

Arguments with no key are assigned to fields by using consecutive integer keys. This functionality is described in more detail in Chapter 5.

By default, fields can contain any type of object, and the objects of different types can be in the same vector. The following is an example of a `grocery_list` that holds the string "apple", and two instances of a `grocery_item`.

```
<defclass grocery_item/>
<grocery_list
  "apple"
  <grocery_item name="milk" quantity=<gallon 2/> />
  <grocery_item name="apple" quantity=2 />
/>
```

For the second value, named "milk", the quantity is two gallons, shown as `<gallon 2/>`. The "apple" item has a quantity of integer 2. You can't tell what unit the quantity is in because the value is the integer 2.

Summary

This chapter described how to create both simple and complex Water objects. You can create a simple instance from the `thing` or `vector` class. Objects can be nested to any depth in the fields of other objects. Water has classes and instances, but with an interesting twist—every object can play the role of both a class and an instance. Water includes a hypertext class that defines all the standard XHTML tags as Water objects.