

## Chapter 2

# ConciseXML Syntax and XML 1.0

### IN THIS CHAPTER

- ◆ Understanding how XML 1.0 syntax was not designed for data
- ◆ Understanding the ambiguity of XML 1.0 representations
- ◆ Understanding how ConciseXML fixes limitations of XML 1.0

THIS CHAPTER COVERS SYNTAX AND DATA representation in XML. XML has some constraints that make it difficult to represent data. The limitations imposed by XML are hard to understand because most articles and books on XML never address these limitations. You can successfully program Water without reading this chapter, but if you really want to understand XML at a fundamental level, this chapter is essential.

## The World Used to be Flat

I believe a good analogy to the problem is how the world went from “flat” to “round”. Hundreds of years ago the prevailing view was the world was flat. That isn’t too hard to believe because we all experience a flat world. Many problems surfaced, though, due to a “flat-world” model. People kept inventing corrections to account for the flawed model, but the complexity due to “fixing” the model became a huge problem. Even the people who saw the complexity of the flat world tried to justify their approach after the elegance of the round-world view was quite clear.

Imagine the difficulty Galileo had trying to convince people that the world was round. As with most revolutionary ideas, people thought he was crazy. How is this discussion about a flat versus a round world related to XML?

The vast majority of people working with XML today believe in the equivalent of an XML flat world. This flat-world view is everywhere and is supported by standards, committees, books, and seminars – it is the dominant view for XML.

The XML flat-world model treats XML as a syntax for documents. The round-world model treats XML as a syntax for data and objects. The differences are slight, but the implications are huge.

A small but growing group of people understands that the flat-world (document-centric) model has outlived its usefulness, and that a round-world (data-centric)

model is required for making progress. This chapter will hopefully be a key tool for converting everyone to a simpler, round-world view. The implications are tremendous, leading you toward seeing XML in a whole new way.

## XML is a Syntax

XML has a lot of misconceptions. Many articles say that XML is either: 1) a standard for describing data in a machine-understandable way, 2) the solution for application integration problems, or 3) a programming language. XML is none of these things. XML is a syntax.

What does this mean? It means that the XML 1.0 standard does not define any tag names or attributes that carry any meaning. XML only describes the syntax of how elements and attributes are represented. XML doesn't even specify a standard way to represent objects and data, because there is no standard for how elements and attributes should be used to represent objects and fields.

Hundreds of standards, such as XHTML, WSDL, UDDI, SVG, and ebXML, use an XML syntax for describing data. Because there is no single way to represent data in XML, every standard chooses its own way to represent data. This has a disastrous effect because each standard with an XML syntax can not easily share data. This problem is not well understood in the industry and has led to significant complexity when trying to deploy solutions based on XML.

## Document versus Data

XML is commonly used to represent data structures. A data structure is just a way to represent data that obeys some well-defined structure. I will describe how Water can formally describe the structure of data by using Water Type and Water Contract. But this chapter shows how to unambiguously represent static data by using Water.

Representing static data might seem straightforward, but XML 1.0 has some design constraints carried over from the document markup world that make representing data in XML quite confusing. A discussion about elements versus attributes is a common example of this confusion.

In most programming languages and other technologies for representing data, there is a concept of a data structure, data value, or object. This book, by convention, will use the term *object*. The word object will be similar to other terms such as a *record*, *structure*, or *tuple* from other technologies.

In most programming languages, an object has fields, and those fields hold values that are other objects. Water objects have this property as well. An object is a collection of fields. Each field has a key and a value. The value can be any object.

The following is an example of an `item` object.

```
<item id="XL283" color="blue" size=10/>
```

The preceding XML could be verbally described as creating an instance of an `item` object. The instance has three fields: `id`, `color`, and `size`. The value of the `id` field is the string "XL283", the value of the `color` field is "blue", and the value of the `size` field is the number 10.

The *type* or *class* of the object appears as the element's name, immediately following the opening angle bracket (<). The fields of the object are represented as key-value pairs within the element's opening area.

When you see an opening angle bracket, it syntactically is the start of an XML element, but it has the semantic meaning of performing a *call*. The call is either the calling of method, or the calling of a constructor method of an object. Fields of an object have a clear and unambiguous *key* and *value*.

```
<item id="xx283" color="blue" size=10/>
```

In the preceding line, the instance of `item` has three fields. "`id`" is the key of the first field, and "xx283" is the value of the field. "`color`" is the key of the second field and "blue" is its value. "`size`" is the key of the third field and the integer 10 is its value.

It is very common, though, to see the following XML to represent the instance of `item` above.

```
<item>
  <id>xx283</id>
  <color>blue</color>
  <size>10</size>
</item>
```

To the vast majority of people, the above XML looks very normal and easily understood, but this is an example of XML in the flat-world model. The round-world model sees this as an ambiguous, poorly constructed XML data object. One problem that is described in detail later in this chapter is that the syntax of an XML element is used to represent two very different things: an object and a field of an object. Having one syntax to represent two different concepts presents a serious ambiguity. This ambiguity leads to a serious problem when a machine tries to interpret the meaning of the XML data.

For a data structure to be useful, the distinction between objects and fields is extremely important. How, for example, do you know that `<color>blue</color>` represents a field of `item` and not an instance of type `color`? As humans, we use our gift of pattern recognition to deduce that `color` must be a field of `item` because it occurs within the content of `item` and it has `blue` in the content of the element.

To emphasize the ambiguity, what if you wrapped the `item` within another `color` element? Is `item` now a field of `color`? Did the meaning of `item` radically change because it moved to a different level in the structure?

```
<color>
  <item>
    <id>xx283</id>
    <color>blue</color>
    <size>10</size>
  </item>
</color>
```

If a serious ambiguity appears in such a small example, imagine the scope of the problem when objects and data structures get more complex. At a minimum, data structures need to be unambiguous and not depend on any other knowledge for interpreting a data structure.



Most XML examples today exhibit the problem of element ambiguity where elements are used for both representing a field of an object as well as objects themselves. The problem occurs in most XML standards and textbook examples from major publishers. It is so common, in fact, that it is almost impossible to find XML examples that do not have this problem. I believe that the “**object-field ambiguity**” of XML is one of the primary reasons why XML is much more complex than necessary. This widespread problem is one of the reasons for the slow pace of XML adoption.

---

Water’s use of XML makes a clear separation between objects and fields. An XML element represents an object. XML attributes represent fields of an object. The ConciseXML syntax allows any type of object as the value of an attribute; therefore, Water supports fields that can store any type of object – not just strings.

## ConciseXML and XML 1.0

XML was derived from SGML and contains some constructs from a document-centric, not data-centric, world. The XML 1.0 syntax has no standard representation for objects with fields. Field key, data, and type information might be encoded in a dozen different ways in a dozen different standards. The majority of developers we talk to find XML 1.0 syntax verbose and cumbersome and avoid using it for many cases. ConciseXML is efficient and readable without the use of special tools. ConciseXML can be about as concise as the comma separated value (CSV) format for data. When used for programming, ConciseXML is about as concise as Java or C++ syntax.

ConciseXML has one consistent representation for encoding data: ConciseXML uses attributes for fields. XML 1.0 specifies that attributes must be quoted strings, but ConciseXML lets the value of a field be any object, not just a string. The key of

a field can also be any object, not just a string. This simple extension significantly reduces the ambiguity of XML 1.0 documents and makes XML much more suited for representing data and logic.

ConciseXML is a superset of XML 1.0 and is both forward and backward compatible with XML 1.0. ConciseXML supports both the XML 1.0 form as well as a concise form. ConciseXML supports three syntactic extensions to XML 1.0 that reduce the verbosity of XML 1.0.

First, ConciseXML permits the closing tag name to be omitted. The ending tag name is redundant because a machine reading the XML can simply match opening tags with closing tags. Requiring the ending tag name encourages developers to choose short, abbreviated tag names. When calling a method in a normal programming language, you don't expect to type the method name twice.

```
ConciseXML: <textarea>hi</>
XML 1.0: <textarea>hi</textarea>
```

An optional closing tag name is actually required for Water because of its dynamism. In some cases, the object to call is calculated during runtime and therefore is not known when writing the code.

```
<set some_text="testing"/>
<set text_input=
  <if> some_text.<more 30/>  hypertext.TEXTAREA
    else                    hypertext.INPUT
  </if>
/>
<text_input 0=some_text/>
```

Second, ConciseXML supports by-position arguments. This means the attribute key can be omitted. When calling a method in a typical programming language, you don't expect to name the arguments in a call. There is no ambiguity because the definition of the method defines the exact ordering of arguments. In this respect, ConciseXML supports the calling convention of popular programming languages.

```
ConciseXML: <person "Mike" "Plusch"/>
XML 1.0: <person first="Mike" last="Plusch"/>
```

Third, ConciseXML supports a dot notation for specifying a path. The dot notation is used in many languages including Java and JavaScript. Dot notation often makes the code more readable, and can significantly reduce the nesting of calls.

ConciseXML:

```
"abe".<foo color="blue"/>.bar
```

XML 1.0:

```
<execute_path>
  "abe"
  <foo color="blue"/>
  bar
</execute_path>
```

ConciseXML also eliminates the need for character entities and replaces them with a standard object reference. See the [www.waterlang.org](http://www.waterlang.org) site for more details.

For every ConciseXML extension to XML 1.0, there is a corresponding representation in the XML 1.0 syntax. A single file can include both ConciseXML and XML 1.0 forms. There could be many different XML 1.0 representations for ConciseXML, but ConciseXML uses a simple representation where attributes that have non-string keys or values are put within an attributes element. The following two examples create the same object:

ConciseXML:

```
<list <item name="bread"/> <item name="milk"/> />
```

XML 1.0:

```
<list>
  <attributes><item name="bread"/><item name="milk"/></attributes>
</list>
```

## Summary

ConciseXML is an innovation on XML 1.0 that enables XML to better handle data and logic. The constraints of XML 1.0 have limited its use because of significant ambiguity and verbosity. ConciseXML removes those constraints and provides a backward-compatible solution.

The constraints of the XML 1.0 syntax have limited its use because of significant ambiguity and verbosity. ConciseXML is an innovation on the XML 1.0 syntax that enables XML to better handle data and logic. ConciseXML removes those constraints and provides a backward-compatible solution.