B

XML Primer

This appendix introduces the Extensible Markup Language (XML) [79]. XML is a text-based language for representing data or documents containing structured information. Similar to the Hyper Text Markup Language (HTML) used to write Web pages, XML documents comprise tags or elements (words bracketed by '$<$' and '$>$') and attributes (of the form name="value"). XML itself does not define the meaning or semantics associated with particular elements or attributes (these are defined in separate specifications). XML merely supplies a set of rules or conventions for designing a document format and also specifies how machines parse such documents.

XML appears throughout this book. The data representation formats used within the MRCP protocol are based on XML. Some examples include the Speech Recognition Grammar Specification (SRGS) for specifying the words and phrases a speech recogniser looks for in an audio stream, the Speech Synthesis Markup Language (SSML) for instructing speech synthesisers how to convert written language input into spoken language output, and the Pronunciation Lexicon Specification (PLS) for defining pronunciation lexicons or dictionaries for speech recognisers and synthesisers. VoiceXML, a language for programming speech applications, is also as its name suggests based on XML.

B.1 Background

XML is derived from a restricted and simplified form of the Standard Generalized Markup Language (SGML). The first version of XML was developed by the XML Working Group at the World Wide Web Consortium (W3C) in 1996. XML is a simple format for adding markup to documents or data that are transported over networks. XML is used as the base format for many diverse applications including vector graphics, Web pages, mathematical equations, distributed object-oriented communication, and interactive voice response dialogs to name but a few. XML is designed to be machine readable and writeable and since it is text-based, it is also readable and writeable by humans. The term XML is also frequently used to refer to a family of related technologies. For example, XSLT is a technology for transforming XML documents into other formats, XLink is a standardised way of adding hyperlinks to XML, and

DOM is a standard set of function calls for manipulating XML documents. The industry has spawned a plethora of XML tools such as parsers, editors, validators and XSLT processors.

B.2 Basic Concepts

XML documents comprise character data and markup. Markup includes: an XML declaration, start-tags, end-tags, empty-element tags, comments, character references, entity references, CDATA sections, processing instructions, and document type declarations (discussed in Section B.4). Let's jump straight in and study an imaginary XML document for describing music radio stations. This document describes a collection of stations and each station contains structured data including its name, the frequency on which it broadcasts, and the music genre. An example is illustrated below:

```
<?xml version="1.0" encoding="UTF-8"?>
<stations>
    <station>
        <name>Classical Hits</name>
        <freq band="FM">101.3</freq>
        <genre>classical</genre>
    </station>
    <station>
        <name>Eclectic Mix</name>
        <freq band="FM">99.4</freq>
        <genre/>
    </station>
    <!-- Add more station elements here -->
</stations>
```

The first thing to notice is that the document commences with the recommended *XML declaration*. The XML declaration identifies the document as XML, includes the XML version number, and optionally specifies the character encoding for the content - in this case UTF-8. The Unicode standard[1] defines codes for characters used in all major languages written today. Character encodings define the identity of each character, its numeric value (called a code point), and the algorithm for mapping each code point into a unique byte sequence. XML processors are required to support the UTF-16 and UTF-8 character encodings. UTF-16 is a 2-byte or 4-byte encoding - the 2-byte encoding is capable of representing the complete Unicode multilingual plane and the 4-byte encoding is capable of representing all planes. UTF-8 is a variable-length character encoding that uses 1 to 4 bytes depending on the character to be encoded. UTF-8 is capable of representing all Unicode characters and is also backward compatible with the ASCII character set (the ASCII characters have the same byte values in UTF-8).

Next the document's root element, `<stations>`, encloses several descendent elements. This document is well formed as required by XML: each *start-tag* has a corresponding *end-tag* (e.g. `<name>` and `</name>`) or is an *empty-element* (e.g. `<genre/>`). *Attributes* are

[1]The Unicode standard is fully compatible and synchronised with the corresponding versions of the International Standard ISO/IEC 10646.

used to associate name-value pairs with elements (e.g. `band="FM"`). Comments are written in XML by enclosing them in the `<!--` and `-->` sequence of characters.

Documents may also contain *character references* for specifying reserved characters or characters not directly accessible from available input devices. A character can be inserted by writing `&#x` followed by the character's code point in ISO/IEC 10646 and terminated with a semicolon. For example, the letter 'A' can be inserted with the code `&#x41;`. Closely related to character references are entity references, which refer to the content of a named entity. Examples of common entity references that are used to escape characters that would otherwise confuse XML parsers include `&lt;` for '<', `&gt;` for '>', and `&amp;` for '&'. Sometimes large blocks of text need to be inserted into an XML document that may contain characters that might be inadvertently recognised as markup. *CDATA sections* can be used to escape a block of data. CDATA sections begin with the string `<![CDATA[` and end with the string `]]>`. For example,

```
<![CDATA[Eclectic <Mix>]]>
```

Finally, *processing instructions* are not part of the document's character data but are used to pass instructions through to the processor. Processing instructions begin with the characters `<?` and end with the characters `?>`, for example `<?Example?>`. Processing instructions are rarely used in practice.

B.3 Namespaces

One of the nice advantages of XML is that it is modular. A new document format can be defined by combining and reusing other formats. Since two formats developed independently may have elements or attributes with the same name, care must be taken when combining those formats (does `<p>` mean paragraph from this format or person from that one?). XML provides a namespace mechanism [80] to qualify element and attribute names by associating them with namespaces identified by URI references (URIs resolve naming collisions because they are assumed to be unique).

Using namespaces requires two steps: the first is to declare the namespace and the second is to apply the namespace to elements and attributes. A namespace is declared using the reserved attribute `xmlns` or prefix `xmlns:`. The following example declares a namespace whose *namespace prefix* is `ex` and whose *namespace name* is the URI reference `http://example.com/name`:

```
<x xmlns:ex="http://example.com/name">
    ...
</x>
```

The `xmlns` attribute can be thought of as a command for relating the letters comprising the prefix (in this case `ex`) to a URI. The URI itself is not accessed and does not need to exist even (it is just employed as a unique string). Note that the previous example declared a namespace on the element `<x>` but that element is not changed in any way by this operation. Further, namespace declarations can appear on any element and multiple declarations may be specified on the same element. If the same namespace prefix is redefined later in the document, the new namespace name (URI reference) overrides the previous one.

The next step is to apply the namespace to elements and attributes. The namespace can be applied to elements and attributes by using the namespace prefix, for example:

```
<ex:x xmlns:ex="http://example.com/name">
    <ex:y>0</ex:y>
    <ex:z ex:a="0">2</ex:z>
</ex:x>
```

In fact, applying the namespace to the attribute `a` is not really necessary since the attribute is uniquely identified by the element it is on. Thus, we could also have written:

```
<ex:x xmlns:ex="http://example.com/name">
   <ex:y>0</ex:y>
   <ex:z a="0">2</ex:z>
</ex:x>
```

A default namespace can be specified for the element in which the namespace is declared, and all its descendants, by dropping the prefix. This document is equivalent to the previous one:

```
<x xmlns="http://example.com/name">
    <y>0</y>
    <z a="0">2</z>
</x>
```

One subtlety: the default namespace only applies to elements - not attributes. In this example, the attribute `a` is not bound to a namespace. So while this example is equivalent to the previous, it is not equivalent to the one before that (which did apply a namespace to the attribute).

As a final example, we illustrate how the notional radio station markup above could be embedded in a HTML host document. In this example, the HTML elements are specified in the default namespace and an explicit namespace prefix is used for the embedded document:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3c.org/1999/xhtml"
       xmlns:s="http://example.com/radio">
    <head>
        <title>Document Embedding Example</title>
    </head>
    <body>
        <p>
            <s:stations>
                <s:station>
                    <s:name>Classical Hits</s:name>
                    <s:freq s:band="FM">101.3</s:freq>
                    <s:genre>classical</s:genre>
                </s:station>
            </s:stations>
        </p>
    </body>
</html>
```

Note that namespaces solve the syntactic problem of how multiple formats may be combined; the actual meaning or semantics of such a document combination must be specified elsewhere.

B.4 Document Schemas

There are three popular ways for enforcing syntactic constraints or schema in an XML document (i.e. how you go about specifying the names of allowable elements and attributes, which attributes appear on which elements,their range of values, which elements may appear as children of other elements, etc.). The *Document Type Definition* (DTD) language is native to the XML specification [79] but has relatively limited expressive power. As a result of the DTD limitations, the W3C's XML Working Group has since created *XML Schema* [33], which delivers a more rigorous and comprehensive language for expressing document structure, attributes, data typing, and so on. Unlike DTDs, the schema is itself written in XML. XML Schema is widely implemented in many XML tools and is used by all the specifications in Part III of this book to express the document's constraints. While XML Schema is exceptionally powerful, it is also somewhat complex to understand and use. A third alternative is the *RELAX NG* [34] specification developed by Oasis. RELAX NG is quite simple to learn and use yet still retains (and in some cases goes beyond) the flexibility afforded by XML Schema.

We conclude this section by looking at how an instance XML document can indicate an associated DTD, XML Schema, or RELAX NG schema. Due to space constraints, we do not discuss the details of the schema languages themselves, rather the interested reader is referred to [79], [33] and [34] respectively. Associating a schema to an XML instance document allows XML tools to validate the document's structure. The XML *document type declaration* is used either to contain or (more usually) to point to a declaration that provides a schema for the document. The document type declaration is indicated via the `DOCTYPE` processing instruction as the following example illustrates:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE EXAMPLE SYSTEM "http://example.com/example.dtd">
<example xmlns="http://example.com">
    <name>Dave</name>
</example>
```

The name of the root element appears in the `DOCTYPE` line followed by the token `SYSTEM` and the location of the DTD file itself. A `PUBLIC` identifier may be used instead of a `SYSTEM` identifier to specify a formal identifier for the DTD, e.g. for HTML:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
            "http://www.w3.org/TR/html4/strict.dtd">
<html>
    ...
</html>
```

An XML schema can be associated with an instance document by providing a hint to the location of the schema in the document as the following example demonstrates:

```
<?xml version="1.0" encoding="UTF-8"?>
<example xmlns="http://example.com"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema-instance"
         xsd:schemaLocation="http://example.com
           http://example.com/schemas/example.xsd">
    <name>Dave</name>
</example>
```

The hint for the location of the schema is specified by the `xsd:schemaLocation` attribute, whose value comprises the namespace of the document in question followed by a location for the schema (the prefix `xsd:` is used by convention to denote the XML Schema namespace, although any prefix can be used).

Finally, the RELAX NG does not provide a way to associate an instance document with its schema. Rather, RELAX NG views this as a problem of associating validation processing with a document that must be performed by the processing tool.