XML General Technical Articles

Understanding Infosets

Martin Gudgin
Microsoft Corporation

July 2002 (Revised February 2004)

Applies to:
Extensible Markup Language (XML) 1.0
XML namespaces specifications

Summary: Covers the most important types of information items and their properties, the mapping between Infoset properties and the serialization format defined by the Extensible Markup Language (XML) 1.0 and Namespaces in XML specifications, and the relationship between the Infoset and XML programming APIs. (10 printed pages)

Contents

Introduction

The XML Information Set (Infoset) defines a data model for XML. This data model is a set of abstractions that detail the properties of XML trees. These abstractions provide a common viewpoint from which to think about XML APIs and higher-level specifications such as XPath, XSLT and XML Schema, as shown in Figure 1.

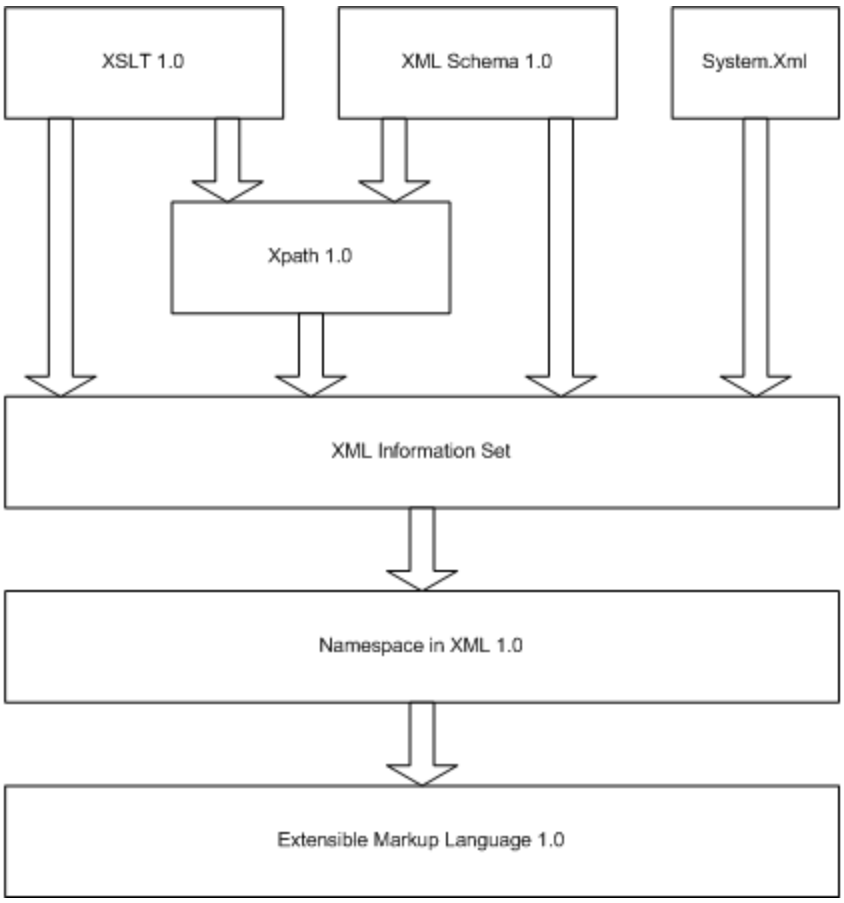

Figure 1. XML abstractions

The W3C XML Information Set recommendation describes an abstract representation of an XML document. The XML Infoset is primarily meant to act as a set of definitions used by XML technologies to formally describe what parts of an XML document they operate upon. Several W3C XML technologies are described in terms of the XML Infoset, including SOAP 1.2, XML Schema, and XQuery.

Having a data model and the associated abstractions is important; without them, each XML specification or API is forced to invent its own. This would mean that XML developers would have to learn a new data model for each new XML technology, all of which would be similar, but not the same. This was the situation in the early days of XML as evidenced by the existence of the W3C Document Object Model (DOM) [http://www.w3.org/dom/] and the XPath data model [http://www.w3.org/tr/xpath.aspx] . In addition, the presence of a common abstract data model makes it possible to build up a body of knowledge about how to use XML that is not dependent on or explicitly tied to a particular XML specification or API. The value of such an independent body of knowledge is evident in the relational model and its effect on the database world. Like the relational model there is no standard API for programming against information items directly. Instead, you work with them indirectly through other XML technologies.

The XML Infoset is a tree-based hierarchical representation of an XML document. An XML document's information set consists of a number of information items, which are abstract representations of the components of an XML document. There are information items representing the document, its elements, attributes, processing instructions, comments, characters, notations, namespaces, unparsed entities, unexpanded entity references, and the document type declaration.

The XML Infoset defines what should be considered to be significant information in an XML document. For example, the infoset does not distinguish between the two forms of empty element. So the following

Copy Code

```
<test></test>
<test/>
```

are considered equivalent according to the XML Infoset. Similarly, the kind of quotation marks used for attributes is not considered significant; thus, the elements

Copy Code

```
<test attr='value'/>
<test attr="value"/>
```

are considered equivalent according to the XML Infoset. A list of aspects of XML 1.0 syntax that are not considered significant by the XML Infoset is provided in Appendix D of the W3C XML Information Set recommendation.

This article examines the more important types of information items and their properties. For an exhaustive list of information items and all their properties please refer to XML Information Set [http://www.w3.org/tr/xml-infoset/] page on the W3C Web site. The article also looks at the mapping between Infoset properties and the serialization format defined by the Extensible Markup Language (XML) 1.0 (Second Edition) [http://www.w3.org/tr/rec-xml.html] and Namespaces in XML [http://www.w3.org/tr/rec-xml-names/] specifications. Finally, the article touches on the relationship between the Infoset and XML programming APIs.

The Root of the Tree

The root of an Infoset is always a document information item. The most important property of the document information item is the **[children]** property, which is a list of information items, in document order (a depth-first traversal of the tree), that are immediate descendants of the document information item. Exactly one of those information items is an element information item. This accurately models the constraint that XML trees must always have exactly one top-level element. This element is known as the document element and is also present in the **[document element]** property of the document information item. Comment and processing instruction information items may also appear in the **[children]** property. In the case where the only child of the document information item is the document element, the **[children]** and **[document element]** properties essentially have the same value.

It is worth noting that without the document information item, XML would be a forest rather than a tree, due to the possibility of comments and processing instructions appearing before and after the document element. It is the document information item, which is pretty much entirely abstract, that makes XML a tree. All other information items in the tree are descendants of the document information item.

Elements and Attributes

XML Information Sets are predominantly made up of element information items and attribute information items. The former can be used to model structured or simple values while the latter can only be used to model simple values. Both types of information item have several important properties. For element information items, the **[parent]** property contains the information item that is the parent of the element information item. In the case of the document element, this will be the document information item, while for all other elements it will be another element information item. In both cases the **[children]** property of the parent information item will contain the element information item. Figure 2 shows an element information item with document information item parent. Note that the structural relationship is bi-directional.

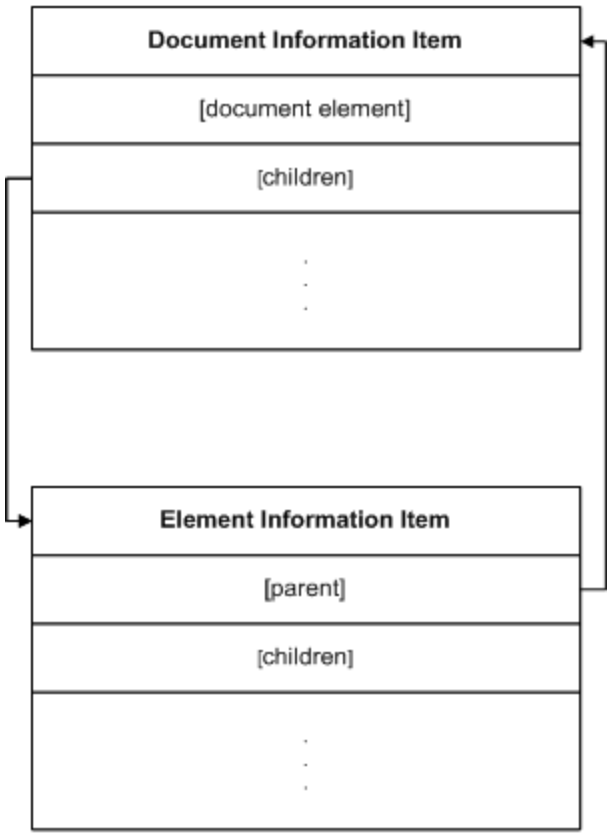

Figure 2. Document information item and document element information item

Element information items also have a **[children]** property that, like the **[children]** property of the document information item, contains all the immediate descendants of the element information item. These descendants may be other element information items, comment or processing instruction information items, or character information items.

Element information items have two name-related properties: **[local name]** and **[namespace name]**. The value of the latter is either the URI of the namespace the element belongs to or, if the element is not associated with a namespace, it is empty. The value of the **[local name]** property is the local part of the name of the element, scoped by the **[namespace name]** property. Together these two properties make up the name of the element information item. Two element information items with the same value for **[local name]** but different values for **[namespace name]** are assumed to be semantically distinct.

The list of attribute information items associated with an element information item is modeled by the **[attributes]** property. This property is an unordered list of attribute information items. Such information items are not considered to be part of the tree directly, so doing a depth-first traversal of the **[children]** properties of all information items starting with the document information item will not encounter any attribute information items.

Attribute information items share some properties with element information items. For instance, the **[namespace name]** and **[local name]** properties of an attribute information item are interpreted the same way as those of an element information item. However, other aspects of attribute information items are modeled differently. There is no **[children]** property; rather, attribute information items have a **[normalized value]** property that contains the simple value of the attribute. Similarly there is no **[parent]** property; instead, there is an **[owner element]** property whose value is the element information item whose **[attributes]** property has the attribute information item in its list. Figure 3 shows an element information item and associated attribute information item.

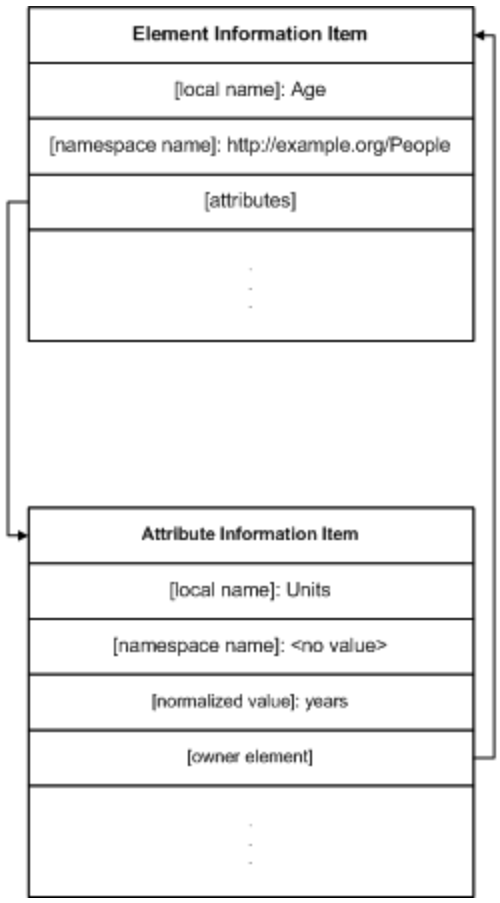

Figure 3. Element information item and associated attribute information item

Characters

Textual content of element information items is modeled using character information items. There is a **[parent]** property that contains the element information item whose **[children]** property contains the character information item. The other important property is the **[character code]** property, the value of which is the ISO 10646 character code that represents the character. Table 1 shows several ISO 10646 characters along with their corresponding names and character codes, while Figure 4 shows an element information item with an associated attribute and character information item children.

Table 1. Characters and character codes

Character	Name	Code
a	Latin Small Letter A	0x0061
ã	Latin Small Letter A with Tilde	0x00E3
æ	Latin Small Letter Ae	0x00E6
α	Greek Small Letter Alpha	0x03B1

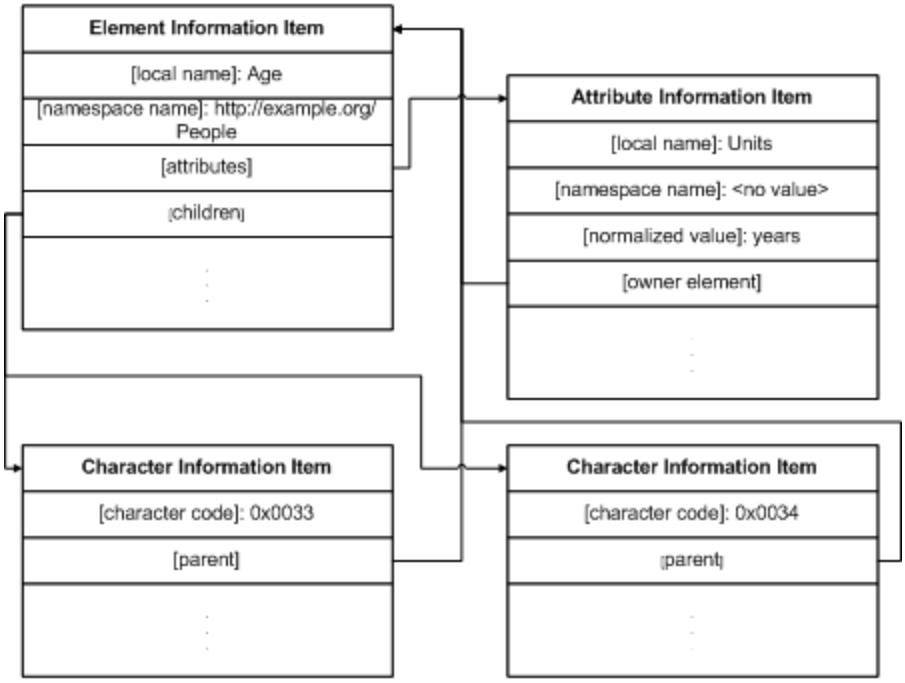

Figure 4. Element information item with an associated attribute and character information item children

Namespaces

Namespaces are used to disambiguate the names of element and attribute information items. As discussed earlier, every element and attribute information item has a two-part name. Where the **[namespace name]** part of that name is not empty, the name is said to be a qualified name, the local part is qualified by the namespace part. The Infoset models namespaces using a combination of namespace information items and attribute information items.

Each element has an **[in-scope namespaces]** property that contains a set of namespace information items. These information items have two properties: a **[namespace name]** property whose value is a URI and a **[prefix]** property whose value is the prefix to which the namespace URI is bound. This prefix to namespace name mapping can be used to determine the namespace affiliation of values in the tree. Such values may appear as the character information item children of element information items, or as the value of attribute information items. For example, Figure 5 shows an attribute whose value refers to the short type in the http://www.w3.org/2001/XMLSchema [http://www.w3.org/2001/xmlschema] namespace.

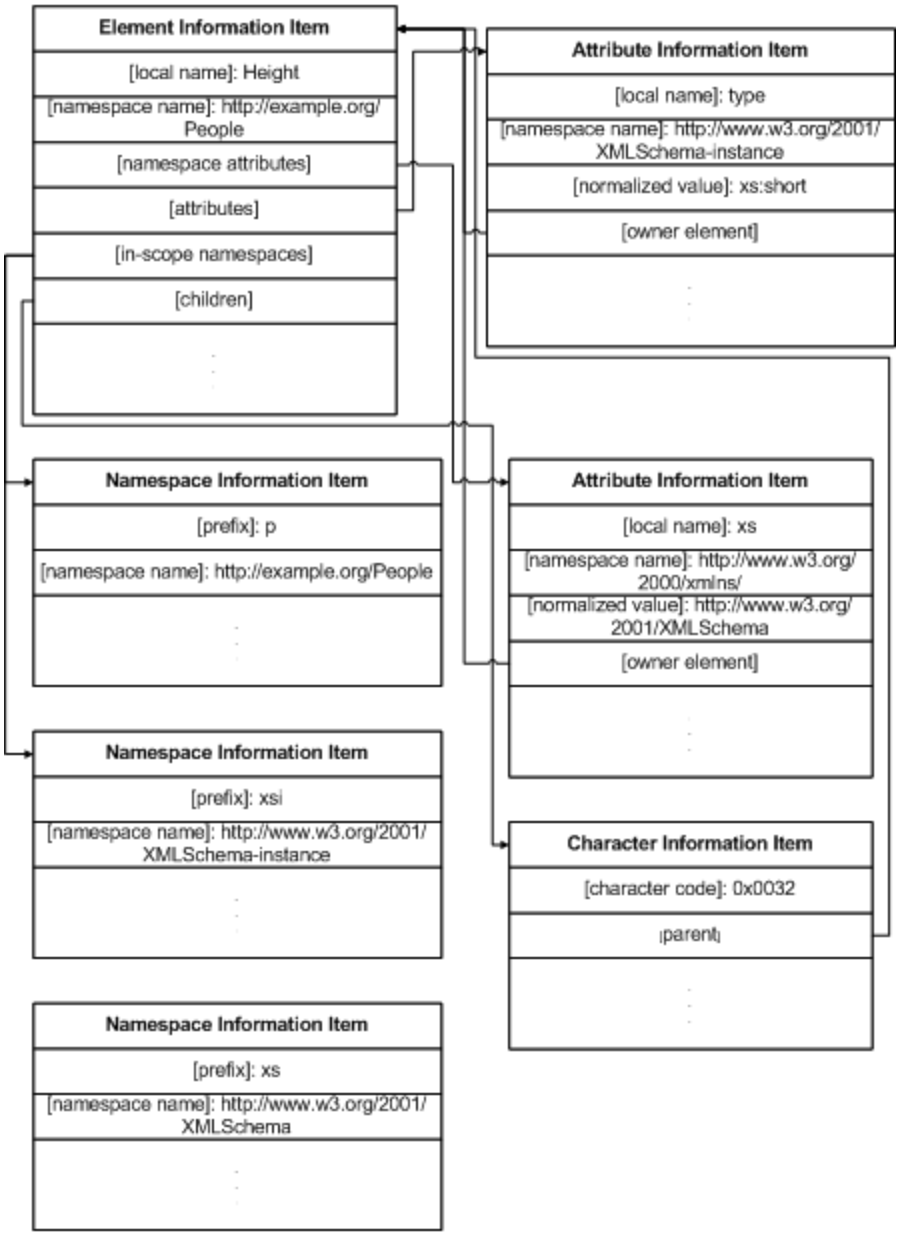

Figure 5. Interpreting an attribute value as part of a namespace

Each element information item also has a **[namespace attributes]** property. This property is a set of attribute information items. For each attribute information item in the set, the **[namespace name]** property is always http://www.w3.org/2000/xmlns/ [http://www.w3.org/2000/xmlns/] . The **[local name]** part is the prefix used when referring to things in that namespace. These attribute information items are important when it comes to serializing and deserializing an Infoset.

Serialization

XML Infosets are typically serialized using the serialization rules laid down by XML 1.0 and Namespaces. Between them, these two specifications detail how all the information items discussed so far are serialized

for storage or transmission.

For example, element information items are serialized using a start tag and an end tag. These tags both contain the **[local name]** property of the element along with a prefix that maps to the **[namespace name]** property. Associated attribute information items including those found in the **[namespace attributes]** property are serialized as name/value pairs inside the start tag. Attribute information items in the **[namespace attributes]** property are crucial to serialization. There must be a prefix to URI mapping for each prefix used in the tree, whether in names or values of element and attribute information items. The serialization defines a scope for such prefix to URI declarations. The code below shows an element information item with a **[local name]** of Person and a **[namespace name]** of http://example.org/People. The element information item has an associated attribute information item in its **[namespace attributes]** property providing a mapping of the prefix **p** to the namespace name http://example.org/People. There is a corresponding namespace information item in the **[in-scope namespaces]** property. The **[children]** property of the element information item is empty. If the element information item had descendant elements, they would each have a namespace information item in their **[in-scope namespaces]** property.

Copy Code

```
<p:Person xmlns:p='http://example.org/People' ></p:Person>
```

XML allows a shorthand, serialized form where an element information item has an empty **[children]** property, shown as follows.

Copy Code

```
<p:Person xmlns:p='http://example.org/People' />
```

The content of the **[attributes]** property is serialized in the same way as the **[namespace attributes]** property. The following code shows the serialized form after adding an attribute information item with a **[local name]** of id and an empty **[namespace name]** property, with a **[normalized value]** of p1.

Copy Code

```
<p:Person xmlns:p='http://example.org/People' id='p1' />
```

The following code shows an alternative serialization; note that more white space is present and the values of the attributes are enclosed in quotation marks, rather than apostrophes.

Copy Code

```
<p:Person     xmlns:p = "http://example.org/People"
              id="p1" />
```

The contents of the **[children]** property of an element information item are serialized between the start and end tags. This includes element information item children, character information item children, as well as comment and processing instruction children.

Character information items are serialized as character codes in whatever encoding the serialized form is written, typically UTF-8 or UTF-16. In fact, if either of these encodings is used, the characters are just written out directly as both encodings can represent all of ISO-10646. However, if a more restricted encoding were used, say ISO-8859-1, then certain character information items may need to be serialized using character references. A character reference is the **[character code]** property serialized as decimal or hexadecimal and delimited by an ampersand and a semi-colon. The following code shows an element information item with an element information item child. That element information item itself has character information item children, some of which are serialized as character references.

Copy Code

```
<p:Person xmlns:p='http://example.org/People' id='p1' >
  <p:Name>&#77;artin &#71;udgin</p:Name>
</p:Person>
```

The fact that, at the level of the Infoset, these serialization variations are invisible, is a benefit to both higher-level specifications and API implementers. It allows both to concentrate on the important properties of XML trees rather than the serialization details. For example, XSLT does not have to detail the fact that both syntactic forms for an element information item with an empty **[children]** property are allowed when writing XSLT transforms. Similarly APIs do not need to expose the delimiter for attribute values. A side effect of viewing things at the Infoset level is that it becomes impossible to deal with things the Infoset does not expose. For example, it is not possible to write an XPath expression to find all attributes whose delimiter was an apostrophe.

It is also interesting to observe that, if another serialization that was not based on pointy brackets was used, but it still provided the properties of the Infoset, higher-level specifications like XSLT would still apply and standard XML APIs could still be used.

Mapping to APIs

The Infoset informs API designers, letting them know which properties of XML trees are important and which are not. XML APIs such as DOM, SAX and XmlReader provide access to the Infoset properties of the underlying data to the application. For example, the following code shows extracting the **[namespace name]** and **[local name]** properties of all element information items in an XML tree using XmlReader.

Copy Code

```
XmlReader xr = new XmlTextReader ( "data.xml" );

while (xr.Read())
{
  if ( NodeType.Element == xt.NodeType )
  {
    Console.WriteLine ( "[local name]: {0}\n[namespace name]: {1}", wr.LocalName,
wr.NamespaceName );
  }
}
```

In addition to dealing with the serialization syntax of XML 1.0 and XML namespaces, it is perfectly possible to layer an XML API on top of a data source that is not XML markup. Microsoft® .NET includes XmlTextReader, specifically for traversing serialized XML text streams, but it also includes XmlNodeReader for traversing nodes in an existing DOM. You can write your own XmlReader implementations on top of other data sources as well. People have written implementations for the file system, the registry, comma-delimited text files, and other formats (See The XML Files: Writing XML Providers for Microsoft .NET [http://msdn.microsoft.com/msdnmag/issues/01/09/xml/toc.asp?frame=true] and XmlCsvReader Implementation [http://msdn.microsoft.com/en-us/library/aa302293.aspx]).

Consumers of an XmlReader implementation still deal with the data as XML, even though the underlying data is never serialized using angle brackets. This is a very powerful technique, because higher-level APIs such as XPath, XSLT and XML Schema can still be applied to these 'synthetic' Infosets.

Conclusion

Philosophers have often asked the question, "If a DOM tree sprouts, grows, withers and dies and is never serialized using angle brackets, is it XML?" Similarly, they have wondered, "If a stream pouring into an XmlReader gurgles to life and then dries up in the heat of the sun, but the underlying data is not angle brackets, is it XML?" The answer to both these questions is "It's an XML Infoset." Moving forward, as more layered specifications are built on top of XML, the more you think and work in terms of Infosets, the better. XML started as a markup language, but it has evolved into a platform, the heart of which is not XML 1.0, but the XML Infoset.