

The Structure of Documents and Schemas

CHAPTER

5

IN THIS CHAPTER

- 5.1 XML Documents 58
- 5.2 The XML Information Set 61
- 5.3 Introduction to the PSVI 67
- 5.4 Introduction to Schemas 69
- 5.5 Schema Documents 70

This chapter introduces XML documents and schemas, each from both the concrete and abstract points of view. When talking about documents, “concrete” means the string of characters and “abstract” means the corresponding object-oriented structure that might be constructed by a parser. That is pretty straightforward.

On the other hand, when talking about *schemas*, “concrete” means schema documents (or parts thereof) *ignoring the distinction between concrete schema document and abstract schema document*, while “abstract” means the object-structured class that might be constructed by a schema processor.

This book assumes understanding of the terminology and concepts of (concrete) XML documents as defined in the XML Recommendation. Section 5.1 relates that material to the corresponding terminology and concepts of abstract XML documents as defined in the Infoset Recommendation. Section 5.2 describes in more detail the concepts of abstract XML documents, which are *information sets (infosets)* or *document information items*, depending on your point of view. In this book, abstract documents are document information items.

5.1 XML Documents

XML documents as defined in the XML Recommendation as strings of characters, but the markup within them implies a rich structure. This structure is implied in that recommendation and defined more carefully in the Infoset Recommendation. The latter Recommendation was written after the Namespace Recommendation, so it is “namespace aware”: The data structures it defines are designed to make namespace information directly available as properties of objects.

NOTATION NOTE

The XML Recommendation uses the term “document” solely to mean the concrete document; The Infoset Recommendation uses the terms “infoset” and “document information item” to mean the abstract document. When dealing with schemas, it is often immaterial whether the documents you think of are concrete or abstract.

A concrete XML document is a string of characters that consists of a *prolog*, an *element*, and “*misc*.” (‘Misc’ really is an XML technical term, but you won’t find it much in this book outside of this paragraph.) The prolog is typically an XML Declaration and/or a Document Type Declaration with *misc* intervening, but may be just an empty string. The element is required and is generally the important part of the document—that is where the *data characters* reside. Misc consists of incidental character strings: whitespace, processing instructions, and comments. The element’s structure and content are of primary interest in this book.

An abstract XML document (an XML infoset) is made up of objects—instances of various classes—combined in various ways. These instances are all called “information items” in the Infoset Recommendation. To begin with, the topmost object is a single document information item.

NOTATION NOTE

Abstract document, infoset, and document information item are essentially the same thing. The Infoset Recommendation takes the point of view that the various information items in isolation, linked by properties, as opposed to being part of the document element, so it considers an infoset a collection of information items with links between them. This distinction is at most a matter of how you think of objects and the values of their properties—or it can be thought of as just an implementation detail.

Other information items occur as values of various properties of the document information item or (recursively) properties of those information items in turn.

Any (concrete) element must be in one of two forms. Either it consists simply of an empty-element tag, or it consists of three consecutive character strings concatenated: a start-tag, content, and an end-tag. Of these three, the simplest is the *end-tag*, which is a string consisting of ‘</’, a type name, and ‘>’. A start-tag is a string consisting of ‘<’, a type name, optional attribute value specifications, and a closing ‘>’, with intervening whitespace where needed or desired after the type name. The type name in the start- and end-tags of an element having start- and end-tags must be the same:

Type name: A name used to name an element type.

An empty-element tag is like a start-tag except that the terminating ‘>’ is instead a ‘/>’.

NOTE

A concrete element, thought of as an object, has no properties special to elements. Instead, it has object-valued `propertyMethods`, which return values, but these values must be defined in terms of the base string. The fact that everything devolves back to the base string is what makes it “concrete.”

The content of an element is slightly more complicated. It, too, is a character string (consisting of the concatenation of data characters, complete elements, and character and entity references), which occurs between the start- and end-tags. If the element consists of an empty-element tag, the content is an empty string.

This book uses the following terms to refer to elements and parts thereof (abstract and concrete) and to connect that terminology to definitions in the various Recommendations:

Document element: The outermost element in an XML document.

Element:

1. A character string conforming to the requirements of the XML Recommendation.
2. An element information item.

Attribute specification: A character string conforming to the requirements of the XML Recommendation.

Attribute:

1. An attribute specification.
2. An attribute information item.

Content of an element:

1. The character string between the start- and end-tags of an element. (Special case: The content of an element consisting of an empty-element tag is *a priori* the empty string.)
2. The terms in the value of an element information item's *children* property (a sequence or list of various kinds of information items, especially *element* and *character* information items).

Children of an element:

1. The immediate subelements of an element.
2. The immediate subelements and data characters of an element.
3. The immediate subelements, data characters, and attributes of an element.

Attributes of an element:

1. The attribute specifications found in the start-tag or empty-element tag of the element.
2. The members of the value of the element information item's *attributes* property (a set of attribute information items).

Subelement of an element: An immediate subelement of the element or (recursively) a subelement of one of those immediate subelements.

Immediate subelement of an element: An element information item term in the element information item's *children* property's value (a sequence or list of information items) or a substring of the content of the element, which, when parsed, gives rise to such an element information item.

Data character:

1. A character in an XML document that an XML parser recognizes as data (rather than markup).
2. A character information item in the *children* of an element information item.
3. Such a character information item or a character in the value of an attribute.

Metadata string:

1. A character string recognized as markup but retained in the abstract data structure because it provides information about the abstract structure. (Example: An element's *type name* or the name of an attribute.)
2. A character string that is the value of an information item property other than the *value* property of an attribute.

Markup punctuation: A character string recognized as markup but which only serves to identify or delineate markup. (Examples: Whitespace and the strings '<', '</', '=', '/>', and '>' found in various tags.) Markup punctuation is typically not retained in the abstract data structure.

5.2 The XML Information Set

The XML Recommendation does not specify the information that a parser should deliver. That has been rectified, at least in part, by the Infoset Recommendation, which states:

This specification defines an abstract data set called the XML Information Set (infoset). Its purpose is to provide a consistent set of definitions for use in other specifications that need to refer to the information in a well-formed XML document.

Conforming XML parsers are not *required* to implement exactly this information set, although, fortunately, many do so in one way or another. Some XML processors actually keep more detailed information about the data in an XML document than is officially in the XML infoset. Indeed, the Infoset Recommendation has an appendix listing a number of things you might expect a Recommendation to have that it does not have. In any case, the infoset is the generic abstraction used by most XML documents and many processors, so it is a useful “mental model.”

XML infosets are important because the model of a schema processor used in the Schema Recommendation is that of a validator operating on an XML infoset, not on the original concrete XML document.

Each XML infoset is made up of objects—instances of various classes, combined in various ways. They are all called *information items*. To begin with, the topmost object is a document information item. Other information items occur as values of various properties of the document information item, or recursively of properties of those information items in turn.

There are two points of view toward abstract documents: One is the document information item view, which holds that the abstract document is a document information item including all the other information items that are (recursively) values of information item properties. The other is the info set view, which holds that the abstract document is a set of information items, including a single document information item, and that information-item-valued properties' values are pointers to other members of the info set. The Info set Recommendation is written so as to be compatible with either point of view. This book generally takes the document information item view.

This book assumes some familiarity with the information items described in the Info set Recommendation; it does not cover all the details of that set. A reader not at all familiar with XML info sets should consult the Recommendation after reading this section. It is not long and not difficult to read after you have it in its proper perspective. See

<http://www.w3.org/TR/xml-info set>.

Eleven distinct classes of information items are defined in that Recommendation:

- *Document information item*
- *Element information item*
- *Attribute information item*
- *Character information item*
- *Processing instruction information item*
- *Unexpanded entity preference information item*
- *Comment information item*
- *Document type declaration information item*
- *Unparsed entity information item*
- *Notation information item*
- *Namespace information item*

Because the first four classes are most important, this chapter covers only those classes.

5.2.1 The *Document Information Item Class*

A document information item has a number of properties containing information pertaining to the whole document. Several are not created by a parser but are added by a validator, whose processing is “cascaded” after that of the parser. The document information item has only one property of particular interest for Schema work: the *document element* property. As you might expect from that property's name, its value is always an element information item—specifically, the one that corresponds to the document element.

Some of the properties of a document information item are related to the DTD and will only be present if the DTD is read and the document DTD-validated. These properties are not of interest for schema validation, as opposed to DTD validation. Schema validation's interest in DTDs is only to be sure that the XML infoset produced by the parser was obtained by expanding all parsable entity references.

Other properties are of interest but need not be dealt with to gain an initial understanding of schema validation. These involve processing instruction, comment, and notation information items.

One DTD-related property is of interest. For proper schema processing, all parsable entity references in the document must be resolved; therefore, it helps to know that all entity declarations have been processed. Each document information item has an *all declarations processed* Boolean-valued property. If true, the parser has asserted that it has processed all entity declarations (as well as all other declarations). For schema processing purposes, of course, related DTD validation results are irrelevant.

5.2.2 The *Element Information Item Class*

Element information items are very important for schema processing: Schema validation is directly concerned with the element tree structure. (We stated previously that the primary property of the document information item of interest is the *document element* property, which has an element information item as its value.) Each element information item has the following properties:

- *Namespace name*
- *Local name*
- *Prefix*
- *Children*
- *Attributes*
- *Namespace attributes*
- *In-scope namespaces*
- *Base URI*
- *Parent*

The first three properties have to do with identifying the element's type. Each element type has a name associated with it, consisting of an optional prefix and colon concatenated with a "local" name devoid of colons. The *namespace name*, if present, is the URI that identifies the namespace and is bound to the local namespace *prefix*; the remainder of the name is the *local name*. (A namespace-aware processor or application should use the namespace name, rather than the prefix, to identify the namespace.)

Children and *attributes* are the properties of primary interest. The *children* property has as its value an ordered list (a finite sequence) of information items that are the parsed equivalent of the content of the element. Because attributes are identified by name rather than by position, a list is not appropriate for them. Also, because an attribute can have only one attribute specification in an element's start-tag (or empty-element tag), the *attributes* property has as its value a *set* of attribute information items.

The next three properties (*namespace attributes*, *in-scope namespaces*, *base URI*) are, like several earlier ones, related to namespaces and are discussed in Chapter 3. The last property, *parent*, provides a back-link up the *children* tree, because all element information items except that of the document element are children of another element information item higher in the tree. (The parent of the document element's information item is the document's document information item.)

5.2.3 The Attribute Information Item Class

The *children* and *attributes* properties of element information items are the important ones. Children are just more element information items, and character information items. But the members of the *attributes* property's value are a new kind of object: attribute information items. Attribute information items are objects having eight properties:

- *Namespace name*
- *Local name*
- *Prefix*
- *Normalized value*
- *Specified*
- *Attribute type*
- *References*
- *Owner element*

The first three properties have to do with identifying the attribute; they function much the same as the element information item properties having the same names. *Owner element* is a back-pointer to the containing element information item. *Specified* is a Boolean-valued property indicating whether the attribute was actually specified in the start-tag (or empty-element tag) of the element involved, as opposed to being inherited as the default value prescribed in the attribute list declaration of the DTD.

NOTE

Schema processing may add a default value if an attribute is not specified in an XML instance, but the indication that this has happened will be in a PSVI-added property, not in the *specified* property.

The normalized value is not always the string of characters enclosed in quotes in the attribute specification, which is the unnormalized value: Character references must be resolved and parsable general entity references should be, and whitespace must be normalized in accordance with the attribute's structure type.

If the value was not explicitly specified but was obtained as the default value prescribed by the entity's declaration, it must still be normalized to deal with parsable general entity references.

In any case, the resulting normalized value is the string of characters that is the attribute's "actual" value. (As far as DTD processing is concerned, that is—schema processing can further normalize the value. See Section 2.3, on the PSVI.)

NOTE

The attribute information item does *not* retain any information about the prenormalization value of an attribute.

Schema processing will possibly further normalize the attribute value, but the result will be in a PSVI-added property different from the *normalized value* property. If the DTD-prescribed normalization and the schema-prescribed normalization are in conflict, unexpected results could occur, because schema normalization does not (and *cannot*) start over from the unnormalized value.

The *attribute type* property's value is the attribute's structure type as specified in the attribute's declaration in the DTD. If the attribute's structure type requires it to reference something—elements with an ID attribute, unparsed entities, or notations—the value of *references* is a list of the corresponding information items. (There are special rules for cases in which the values specified do not correspond to anything declared in the DTD; see the Infoset Recommendation for details.)

NOTATION NOTE

The phrase “structure type” has already been introduced in Chapter 1. In the XML Recommendation and Infoset Recommendation, this is just called the attribute’s “type.” This book avoids just plain ‘type’ because so many different kinds of types are involved when Schema enters the picture.

NOTE

The Infoset Recommendation does not address the form of the *attribute type* values; they are akin to values in a C or Java enumeration: named things from a finite set that can be distinguished. They are not necessarily character strings consisting of those names.

5.2.4 The *Character Information Item Class*

Element information items and character information items are the two most important terms in an element information item’s *children* sequence. A character information item has only three properties: *character code*, *element content whitespace*, and *parent*. The *character code* is a nonnegative integer, the Unicode code for the character. The *element content whitespace* is a Boolean, necessarily FALSE if the character is not whitespace. For whitespace it may be TRUE, FALSE, UNKNOWN, or NO VALUE (see the Infoset Recommendation for details). The *parent*, of course, is a pointer back to the element information item in whose *children* property sequence the character information item occurs.

5.2.5 XML Information Set Summary

An abstract XML document is a document information item. For Schema purposes, the most important property is its *document element*.

An abstract element is an element information item. For Schema purposes, the important properties are those related to the element’s type name, the *attributes* property, and the *children* property.

The *attributes* property’s value is a set of abstract attributes (attribute information items); the *children* property’s value is a list or sequence of abstract characters (character information items), abstract elements (element information items), and other less important abstract things (information items relating to processing instructions, comments, and so on).

An abstract attribute is an attribute information item. For Schema purposes, the important properties are those related to the attribute’s name and the *normalized value* property.

An abstract character is a character information item. It identifies a character by its Unicode code and indicates whether that character is (known to be) a whitespace character in a position where a DTD does not permit PCDATA.

NOTE

A schema can also determine that a whitespace character is in a position where PCDATA is not permitted; it will indicate this in the PSVI by using a different mechanism from the abstract character property used by DTD processors.

5.3 Introduction to the PSVI

Schema processors do not change any values in the “basic” infoSet. Instead, a schema processor provides values for new properties not defined for the basic infoSet. An infoSet with these additional properties is a PSVI. Chapter 16 describes the additional properties in detail; this section provides only an introduction and a partial description of the more fundamental ones.

By adding additional properties, the PSVI definitions define new classes derived from the old. They are generally known by the same names. PSVI also defines new classes, instances of which are used only as values of PSVI-added properties. For example: There are new classes of information items that mimic all of the schema component classes, so that a PSVI can include as a property value a complete “carbon copy” of the schema or any of its components.

A number of the additional properties, and properties provided by the additional classes, are optional, so a “lightweight” processor can legitimately not provide access to them (or can simply always give ABSENT as the value).

NOTATION NOTE

Both the InfoSet Recommendation and the Schema Recommendation require that properties always have a value. Each also provides a special “designated value” that is the value of an “optional” property when that property “does not have a value.” The InfoSet Recommendation calls this value “NO VALUE.” The Schema Recommendation calls this value “ABSENT,” even when discussing the PSVI additions to the XML infoSet. The divergence was probably accidental, caused by the Schema Recommendation going final while the InfoSet Recommendation was still being developed. This book uses both terms, depending on context, to make correlation with material in the appropriate Recommendation easier. In point of fact, the two terms have the same meaning.

Of the various information item classes of the XML infoset, only element information items and attribute information items provide additional properties in the PSVI.

The PSVI version of element information item is derived from the original by adding 23 additional properties. They are described in more detail in Chapter 16. Seven of these properties are peculiar to element information item; the remaining 16 are also added to attribute information item. Attribute information item gets no extra properties not shared with element information item.

In this section, the following properties are mentioned and briefly discussed:

- *Schema normalized value*
- *Nil*
- *ID/IDREF table*
- *Identity-constraint table*

Of these properties, only the first applies to attributes, the first two apply to all elements, and the last two only apply to selected elements. See Chapter 16 for a more complete discussion of the properties and information items added by the PSVI, and an overview of the process by which those properties are given values.

Most of these properties serve to identify the element type or attribute type and the structure type of the attribute or element.

The *schema normalized value* property provides, for attributes and for elements with simple content, a value normalized according to the schema in much the same way that an attribute's value might be normalized according to an attribute definition in a DTD. Because elements with simple content are much like attributes, it should not be surprising that both attribute and element information items get a *schema normalized value* property. The schema normalized value is obtained by normalizing the string of characters that constitutes the content of the element concerned, with any processing instructions or comments stripped out.

NOTE

Schema processing is *not* required to provide the value from the value space of the datatype of an element with simple content or an attribute—only the normalized lexical representation of that value from the lexical space. No provision for attaching the “real” value is made in the PSVI. However, do not be surprised if vendors of schema processors provide a proprietary way to easily get the real value. Without such a facility, each application would have to access the lexical-space string and the information identifying the datatype, and provide its own conversion routines.

continues

For example, in the float datatype, all values have standard 32-bit representations prescribed by an IEEE standard (IEEE 754). The number “positive zero” has a representation consisting of 32 zero-bits. That number is named in the lexical space by the four-character string ‘+0.0’ (among others). The four-character string may show up as a schema normalized value, but the corresponding 32-bit bit string is not required by the Schema Recommendation to be provided anywhere in the PSVI.

The *nil* property serves to indicate if an element was explicitly nulled (that is, `xsi:nil` specified true as well as empty content).

The *ID/IDREF table* is a collection of new kinds of information items not found in the basic infoset that are of use in checking uniqueness of IDREF references. The process of schema validation is defined in terms of actions on the *ID/IDREF table* and hence either the table must be created or equivalent computations worked out. However, upon completion of the validation process, the validator is *not* required to retain the table in the PSVI it makes visible to the receiving application. The *identity-constraint table* serves a similar role with respect to the expanded identity constraints provided by Schemas. This property is provided only by an element information item and will be absent except on the root element where schema processing is invoked (normally the document element, the root of a document’s element tree).

5.4 Introduction to Schemas

Similar to the infoset and document information item points of view for abstract documents, there are two points of view for Schemas. The schema set point of view is that a schema is a *set* of objects generically called “schema components,” with component-valued properties’ values being pointers among the members of the set; the schema component view is that a schema is a particular kind of schema component and all other components of the schema are (recursively) values of component-valued properties of the components.

The Schema Recommendation generally takes the former point of view and does not include a schema schema component in the schema set; nonetheless, it defines a *schema* schema component and thus implements either point of view. This book generally takes the schema schema component point of view.

A *schema* (a schema schema component) has seven set-valued properties:

- *Type definitions*
- *Attribute declarations*
- *Element declarations*

- *Attribute group definitions*
- *Model group definitions*
- *Notation declarations*
- *Annotations*

These seven sets contain the corresponding top-level schema components; the *type definitions* set contains both simple type and complex type components. All remaining components, including all six types not accounted for in the preceding list, show up as property values of some of these components (recursively, of course—just like information items in a document information item). So a *schema* is an abstract object much like a DTD—if a DTD had ever been defined as an object.

NOTE

Just as a DTD does not prescribe the type of the document element (that is done only by the document type *declaration*), a schema does not prescribe the type of the document element.

The various kinds of schema components are described in Chapter 15.

5.5 Schema Documents

The document element of a schema document is a schema element. Schemas are made up of elements of various other types; there are element types corresponding to the different schema components. An element corresponds to an “element declaration schema component”—a (*Schema-defined*) *element type*. A `simpleType` element contains the information necessary to create a simple type (a “simple type definition schema component”). Details of the use and construction of the various types of elements in a schema document are found throughout Part II of this book.

More than one schema document may contribute to a schema for either or both of these reasons:

- Useful parts of the schema can be separated out as reusable modules.
- Different components of the schema need to be associated with different namespaces. (All of the components arising from a particular schema document must be associated with the same namespace.)

For these reasons, schema documents are able to include other schema documents, using various mechanisms:

- **include**: If a schema document describes part of the desired schema and targets that part to the same namespace as the target namespace of the schema document being written, that schema part can be included by using an `include` element in the schema document being written. An `include` element includes a pointer to the schema document to be included.

The schema processor, while building the schema, will find and process the included schema document, placing the schema components it represents into the new, larger schema being built.

It is also possible for an included schema document to be coerced to the target namespace of the including schema document, if the included part has no target namespace. This can make includable partial-schema documents even more reusable, but some care must be taken.

`include` elements are described in Chapter 7.

- **redefine**: A `redefine` element is like an `include`, except that within it you can force some changes on various parts of the included schema document as it comes in. This is more controllable than simple cut-and-paste when making “one-offs.”
`redefine` elements are described in Chapter 7.
- **import**: An `import` element is again somewhat similar to an `include`, but the imported schema document is expected to have its own target namespace. This is the only mechanism for building a schema with multiple target namespaces.

Chapter 3 provides more information on the use of namespaces with schemas. `import` elements are described in Chapter 7.

