Building XHTML™ Modules

W3C Working Draft 5 January 2000

This version:
: http://www.w3.org/TR/2000/WD-xhtml-building-20000105
: (Single HTML file [p.1] , Postscript version, PDF version, ZIP archive, or Gzip'd TAR archive)

Latest version:
: http://www.w3.org/TR/xhtml-building

Previous version:
: http://www.w3.org/TR/1999/WD-xhtml-building-19990910/

Diff-marked version:
: xhtml-building-diff-20000105.html

Editors:
: Murray Altheim, Sun Microsystems
: Shane McCarron, Applied Testing and Technology

Abstract

This working draft defines the mechanism for defining markup language modules that are compatible with the modularization framework used by XHTML. This includes a definition of the way in which an abstract module is specified, the way in which this abstraction is mapped into an XML DTD, and the way in which the resulting DTD module can be combined with other XHTML DTD modules to create new markup languages. In the future, it is expected that instructions will also be provided for mapping the abstract specifications into an XML Schema [XMLSCHEMA [p.29]]. Note that the materials in this document were formerly part of the Modularization of XHTML document, but have been separated out for editorial purposes.

Status of this document

This section describes the status of this document at the time of its publication. Other documents may supersede this document. The latest status of this document series is maintained at the W3C.

This is the "Last Call Working Draft" of "Building XHTML Modules". The Last Call review period ends at 2359Z on 1 Feburary 2000. Please send review comments before the review period ends to www-html-editor@w3.org.

The Working Group anticipates asking the W3C Director to advance this document to Proposed Recommendation after the Working Group processes Last Call review comments and incorporates resolutions into the Guidelines.

This document has been produced as part of the W3C HTML Activity. The goals of the HTML Working Group *(members only)* are discussed in the HTML Working Group charter *(members only)*.

This is a W3C Working Draft for review by W3C Members and other interested parties. It is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in progress". This is work in progress and does not imply endorsement by, or the consensus of, either W3C or participants of the HTML WG Group.

A list of current W3C Recommendations and other technical documents can be found at http://www.w3.org/TR.

Quick Table of Contents

Full Table of Contents

1. Introduction

This section is *normative*.

1.1. Why Build XHTML Modules?

XHTML is more than just a recasting of HTML into XML. It is also an extensible architecture that permits the ready definition of new document types. The W3C envisions that client manufacturers, document authors, and content providers may all use this architecture to define document types that are specific to their needs. The XHTML Modularization specification defines a collection of modules and a framework that make the definition of these new document types relatively easy.

That architecture by itself may not be sufficient for the needs of all document type creators. In particular, people who are defining new functionality or combining new functionality with existing elements need a way to define that functionality. The XHTML method for doing this is through the definition of an XHTML module.

XHTML modules define elements and their attributes, add attributes to elements defined in other modules, add values to the set of values available to an attribute defined in other modules, define content models, or some combination of these things. The expression of a module is done through the creation of a prose functional description of the module, an abstract definition of the module's contents, and then one or more implementations of the module. The remainder of this document defines the way in which these steps should be conducted.

1.2. Abstract Modules

An XHTML document type is defined as a set of modules. Each XHTML module has an abstract definition that generally indicates the facilities made available through the module and way those facilities are minimally integrated with each other and with an (eventual) document type.

1.3. XML DTD Modules

An XML DTD module consists of a set of element types, a set of attribute list declarations, and a set of content model declarations, where any of these three sets may be empty. An attribute list declaration in an XML DTD module may modify an element type outside the element types in the module, and a content model declaration may modify an element type outside the element type set.

2. Terms and Definitions

This section is *informative*.

While some terms are defined in place, the following definitions are used throughout this document. Familiarity with the W3C XML 1.0 Recommendation [XML] [p.29] is highly recommended.

abstract module
: a unit of document type specification corresponding to a distinct type of content, corresponding to a markup construct reflecting this distinct type.

compound document
: A compound document is a document that uses more than one XML Namespace. Compound documents may be defined as documents that contain elements or attributes from multiple document types.

content model
: the declared markup structure allowed within instances of an element type. XML 1.0 differentiates two types: elements containing only element content (no character data) and mixed content (elements that may contain character data optionally interspersed with child elements). The latter are characterized by a content specification beginning with the "#PCDATA" string (denoting character data).

document model
: the effective structure and constraints of a given document type. The document model constitutes the abstract representation of the physical or semantic structures of a class of documents.

document type
: a class of documents sharing a common abstract structure. The ISO 8879 [SGML] [p.29] definition is as follows: "a class of documents having similar characteristics; for example, journal, article, technical manual, or memo. (4.102)"

document type definition (DTD)
: a formal, machine-readable expression of the XML structure and syntax rules to which a document instance of a specific document type must conform; the schema type used in XML 1.0 to validate conformance of a document instance to its declared document type. The same markup model may be expressed by a variety of DTDs.

driver
: a generally short file used to declare and instantiate the modules of a DTD. A good rule of thumb is that a DTD driver contains no markup declarations that comprise any part of the document model itself.

element
: an instance of an element type.

element type
: the definition of an element, that is, a container for a distinct semantic class of document content.

entity
: an entity is a logical or physical storage unit containing document content. Entities may be composed of parse-able XML markup or character data, or unparsed (ie., non-XML,

possibly non-textual) content. Entity content may be either defined entirely within the document entity ("internal entities") or external to the document entity ("external entities"). In parsed entities, the replacement text may include references to other entities.

entity reference
: a mnemonic or numeric string used as a reference to the content of a declared entity (eg., "&" for "&", "<" for "<", "©" for "©".)

generic identifier
: the name identifying the element type of an element. Also, element type name.

instantiate
: to replace an entity reference with an instance of its declared content.

markup declaration
: a syntactical construct within a DTD declaring an entity or defining a markup structure. Within XML DTDs, there are four specific types: entity declaration defines the binding between a mnemonic symbol and its replacement content. element declaration constrains which element types may occur as descendants within an element. See also content model. attribute definition list declaration defines the set of attributes for a given element type, and may also establish type constraints and default values. notation declaration defines the binding between a notation name and an external identifier referencing the format of an unparsed entity

markup model
: the markup vocabulary (ie., the gamut of element and attribute names, notations, etc.) and grammar (ie., the prescribed use of that vocabulary) as defined by a document type definition (ie., a schema) The markup model is the concrete representation in markup syntax of the document model, and may be defined with varying levels of strict conformity. The same document model may be expressed by a variety of markup models.

module
: an abstract unit within a document model expressed as a DTD fragment, used to consolidate markup declarations to increase the flexibility, modifiability, reuse and understanding of specific logical or semantic structures.

modularization
: an implementation of a modularization model; the process of composing or de-composing a DTD by dividing its markup declarations into units or groups to support specific goals. Modules may or may not exist as separate file entities (ie., the physical and logical structures of a DTD may mirror each other, but there is no such requirement).

modularization model
: the abstract design of the document type definition (DTD) in support of the modularization goals, such as reuse, extensibility, expressiveness, ease of documentation, code size, consistency and intuitiveness of use. It is important to note that a modularization model is only orthogonally related to the document model it describes, so that two very different modularization models may describe the same document type.

parameter
: entity an entity whose scope of use is within the document prolog (ie., the external subset/DTD or internal subset). Parameter entities are disallowed within the document instance.

parent document type
: A parent document type of a compound document is the document type of the root element.

tag
: descriptive markup delimiting the start and end (including its generic identifier and any attributes) of an element.

3. Conformance Definition

This section is *normative*.

In order to ensure that XHTML modules are maximally portable, this specification rigidly defines conformance requirements. While the conformance definitions can be found in this section, they necessarily reference normative text within this document, within the base XHTML specification [XHTML1 [p.29]], and within other related specifications. It is only possible to fully comprehend the conformance requirements of XHTML through a complete reading of all normative references.

3.1. Module Conformance

This specification defines a method for defining XHTML-conforming modules. A module conforms to this specification when it meets all of the following criteria:

1. The module must be defined using one of the implementation methods identified in this specification (currently on XML DTDs are defined).
2. The module must have a unique identifier as defined in Naming Rules [p.11] .
3. When the module is implemented using an XML DTD, the module must insulate its parameter entity names through the use of unique prefixes or other, similar methods.
4. The module must have a prose definition that describes the syntactic and semantic requirements of the elements, attributes, and/or content models that it declares.
5. The module must not reuse any element names that are defined in other W3C-defined modules, except when the content model and semantics of those elements are either identical to the original or an extension of the original.
6. The module's elements and attributes must be part of an XML Namespace [XMLNAMES [p.31]]. If the module is defined by an organization other than the W3C, this namespace must NOT be the same as the namespace in which other W3C modules are defined.

3.2. Naming Rules

Names for XHTML-conforming document types must adhere to strict naming conventions so that it is possible for software and users to readily determine the relationship of document types to XHTML. The names for modules are defined through XML Formal Public Identifiers (FPIs). Within FPIs, fields are separated by double slash character sequences (`//`). The various fields MUST be composed as follows:

1. The leading field identifies the resources relationship to a formal standard. For privately defined resources, this field MUST be "`-`". For formal standards, this field MUST be the formal reference to the standard (e.g. `ISO/IEC 15445:1999`).
2. The second field MUST contain the name of the organization responsible for maintaining the named item. There is no formal registry for these organization names. Each organization SHOULD define a name that is unique. The name used by the W3C is, for example, `W3C`.

3. The third field MUST take the form `ELEMENTS XHTML-` followed by an organization-defined unique identifier (e.g. MyML 1.0). This identifier is SHOULD be composed of a unique name and a version identifier that can be updated as the document type evolves.
4. The fourth field defines the language in which the item is developed (e.g. `EN`).

Using these rules, the name for an XHTML conforming module might be `-//MyCompany//ELEMENTS XHTML-MyModule 1.0//EN`.

3.2.1. Rationale for Naming Rules

Naming Rules are critical for portability of user agents and XHTML-conforming tools. These rules need to be simple enough that they can be readily adhered to, and need to convey upon document type and module designers the power to readily associate their creations with XHTML (for marketing purposes, if nothing else). The above rules address these concerns. There were some other possibilities for naming conventions, and they were not used for the following reasons:

- Use the XHTML version in the identifier.

  In the case of new modules, there is no need to associate the module with a specific version of XHTML - the name does not need to identify version dependencies.

4. Defining Abstract Modules

This section is *normative*.

An Abstract Module is a definition of an XHTML module using prose text and some informal markup conventions. While such a definition is not generally useful in the machine processing of document types, it is critical in helping people understand what is contained in a module. This section defines the way in which XHTML abstract modules are defined. An XHTML conforming module is *not required* to provide an abstract module. However, anyone developing an XHTML module is encouraged to provide an abstraction to ease in the use of that module.

4.1. Syntactic Conventions

The abstract modules are not defined in a formal grammar. However, the definitions do adhere to the following syntactic conventions. These conventions are similar to those of XML DTDs, and should be familiar to XML DTD authors. Each discrete syntactic element can be combined with others to make more complex expressions that conform to the algebra defined here.

element name
: When an element is included in a content model, its explicit name will be listed.

Content set
: Some modules define lists of explicit element names called *content sets*. When a content set is included in a content model, its name will be listed.

`expr ?`
: Zero or one instances of expr are permitted.

`expr +`
: One or more instances or expr are required.

`expr *`
: Zero or more instances of expr are permitted.

`a , b`
: Expression `a` is required, followed by expression `b`.

`a | b`
: Either expression a or expression b is required.

`a - b`
: Expression a is permitted, omitting elements in expression b.

parentheses
: When an expression is contained within parentheses, evaluation of any subexpressions within the parentheses take place before evaluation of expressions outside of the parentheses (starting at the deepest level of nesting first).

extending pre-defined elements
: In some instances, a module adds attributes to an element. In these instances, the element name is followed by an ampersand (`&`).

Defining the type of attribute values
: When a module defines the type of an attribute value, it does so by listing the type in parentheses after the attribute name.

Defining the legal values of attributes
: When a module defines the legal values for an attribute, it does so by listing the explicit legal values (enclosed in quotation marks), separated by verical bars (|), inside of parentheses following the attribute name.

4.2. Content Types

Abstract module definitions define minimal, atomic content models for each module. These minimal content models reference the elements in the module itself. They may also reference elements in other modules upon which the abstract module depends. Finally, the content model in many cases requires that text be permitted as content to one or more elements. In these cases, the symbol used for text is PCDATA. This is a term, defined in the XML 1.0 Recommendation, that refers to processed character data. A content type can also be defined as EMPTY, meaning the element has no content in its minimal content model.

4.3. Attribute Types

In some instances, it is necessary to define the types of attribute values or the explicit set of permitted values for attributes. The following attribute types (defined in the XML 1.0 Recommendation) are used in the definitions of the Abstract Modules:

Attribute Type	Definition
CDATA	Character data
ID	A document-unique identifier
IDREF	A reference to a document-unique identifier
NAME	A name with the same character constraints as ID above
NMTOKEN	A name composed of CDATA characters but no whitespace
NMTOKENS	Multiple names composed of CDATA characters separated by whitespace
PCDATA	Processed character data

4.4. An Example Abstract Module Definition

This section defines a sample abstract module as an example of how to take advantage of the syntax rules defined above. Since this exampple is trying to use all of the various syntactic elements defined, it is pretty complicated. Typical module defintions would be much simpler than this. Finally, note that this module references the attribute collection Common. This is a collection defined in the XHTML Modularization specification that includes all of the basic attributes that most elements need.

4.4.1. XHTML Skiing Module

The XHTML Skiing Module defines markup used when describing aspects of a ski lodge. The elements and attributes defined in this module are:

Elements	Attributes	Minimal Content Model
resort	Common, href (CDATA)	description , Aspen+
lodge	Common	description, (Aspen - lift)+
lift	Common, href	description?
chalet	Common, href	description?
room	Common, href	description?
lobby	Common, href	description?
fireplace	Common, href	description?
description	Common	PCDATA*

This module also defines the content set Aspen with the minimal content model lodge | lift | chalet | room | lobby.

5. XML DTD Modules

This section is *normative.*

5.1. Implementing Document Model Modules in the DTD

Partitioning of the document model occurs at the abstract module level. This partitioning is implemented in the markup model by two primary methods: parameterization, the use of parameter entities as reusable strings, and modularization, the creation of DTD fragments called *modules*.

5.1.1. Parameterization

This specification classifies parameter entities into six categories and names them consistently using the following suffixes:

.mod
: parameter entities use the suffix `.mod` when they are used to represent a DTD module (a collection of element classes). In this specification, each module is an atomic unit and may be represented as a separate file entity.

.module
: parameter entities use the suffix `.module` when they are used to control the inclusion of a DTD module by containing either of the conditional section keywords `INCLUDE` or `IGNORE`.

.content
: parameter entities use the suffix `.content` when they are used to represent the content model of an element type.

.class
: parameter entities use the suffix `.class` when they are used to represent elements of the same class.

.mix
: parameter entities use the suffix `.mix` when they are used to represent a collection of element types from different classes.

.attrib
: parameter entities use the suffix `.attrib` when they are used to represent a group of tokens representing one or more complete attribute specifications within an ATTLIST declaration.

For example, in HTML 4.0, the `%block;` parameter entity is defined to represent the heterogenous collection of element types that are block-level elements. In this specification, the corollary parameter entity is `%Block.mix;`.

5.1.2. Modularization

DTD modules are often used to encompass the markup declarations of a specific semantic component or "feature", from higher-level document features like tables and forms, to lower-level components such as specific elements or element groups. Modules can even contain modules, creating a hierarchical structure mirroring the document model. Note that modules are not always implemented as separate file entities, and modular DTDs can be easily normalized into single file versions for more efficient distribution over the Web.

The relationship between document model components and how they are implemented in markup as modules, entities and files (i.e., the *granularity* of the parameterization or modularization, how the markup model is structured and stored as separate entities, etc.) is not necessarily direct, as design style and implementation issues properly play a part. Higher-level modules are sometimes delivered as individual file entities to facilitate portability and reusability. To promote interoperability, the XHTML DTD design considers each module as atomic, with the notion that implementations should support the semantics of an entire module without further subdivision.

While the notion of "plug and play" with DTD modules is very attractive, in practice this is not quite so simple. Complex document models often resort to extensive parameterization of abstract modules to facilitate understanding, markup reuse, extensibility, and maintenance. The resultant modules may have many interdependencies, and may require a fair amount of "rewiring" when adding or removing a DTD module. In light of this, a compromise must be made between markup flexibility, complexity of the DTD, and ease of maintainability.

The XHTML DTD attempts to ameliorate this by localizing many of the more "global" parameter entities to several sub-modules that are brought in via a "framework" module. These include declarations for common names, attributes, parameter and character entities.

XHTML elements are classified into the following categories:

structural element types
: element types that create the overall structure of an XHTML document.

block element types
: element types that should cause a line break.

inline element types
: element types that are displayed inline to an existing block.

phrasal element types
: element types that specify a domain-relevant connotation

presentational element types
: element types that indicate a desire on the part of the author for a specific rendering effect.

special case (or "feature") element types
: element types that provide XHTML with special features, such as linking, forms, etc.

6. Developing DTDs with defined and extended modules

This section is **informative**.

The primary purpose of defining XHTML modules and a general modularization methodology is to ease the development of document types that are based upon XHTML. These document types may extend XHTML by integrating additional capabilities (e.g. [SMIL] [p.30] or [MathML] [p.29]), or they may define a subset of XHTML for use in a specialized device. Regardless of the application, XHTML modules are up to the task. This section describes the techniques that document type designers must use in order to take advantage of this modularization architecture. It does this by applying the techniques defined in the previous sections in progressively more complex ways, culminating in the creation of a complete document type from disparate modules.

Note that in no case do these examples require the modification of the XHTML-provided module *files* themselves. The XHTML module files are completely parameterized, so that it is possible through separate module definitions and *driver files* to customize the definition and the content model of each element and each element's hierarchy.

Finally, remember that most users of XHTML are *not* expected to be DTD authors. DTD authors are generally people who are defining specialized markup that will improve the readability, simplify the rendering of a document, or ease machine-processing of documents, or they are client designers that need to define the specialized DTD for their specific client. Consider these cases:

- An organization is providing subscriber's information via a web interface. The organization stores its subscriber information in an XML-based database. One way to report that information out from the database to the web is to embed the XML records from the database directly in the XHTML document. While it is possible to merely embed the records, the organization could define a DTD module that describes the records, attach that module to an XHTML DTD, and thereby create a complete DTD for the pages. The organization can then access the data within the new elements via the Document Object Model [DOM] [p.30] , validate the documents, provide style definitions for the elements that cascade using Cascading Style Sheets [CSS2] [p.??] , etc. By taking the time to define the structure of their data and create a DTD using the processes defined in this section, the organization can realize the full benefits of XML.
- An Internet client developer is designing a specialized device. That device will only support a subset of XHTML, and the devices will always access the Internet via a proxy server that is validating content before passing it on to the client (to minimize error handling on the client). In order to ensure that the content is valid, the developer creates a DTD that is a subset of XHTML using the processes defined in this section. They then use the new DTD in their proxy server and in their devices, and also make the DTD available to content developers so that developers can validate their content before making it available. By performing a few simple steps, the client developer can use the architecture defined in this document to greatly ease their DTD development cost *and* ensure that they are fully

supporting the subset of XHTML that they choose to include.

6.1. Defining additional attributes

In some cases, an extension to XHTML can be as simple as additional attributes. Attributes can be added to an element just by specifying an additional ATTLIST for the element, for example:

```
<!ATTLIST a
      myml:myattr   CDATA        #IMPLIED
>
```

would add the "myattr" attribute, in the "myml" namespace, with a value type of CDATA, to the "a" element. This works because XML permits the definition or extension of the attribute list for an element at any point in a DTD.

Naturally, adding an attribute to a DTD does not mean that any new behavior is defined for arbitrary clients. However, a content developer could use an extra attribute to store information that is accessed by associated scripts via the Document Object Model (for example).

6.2. Defining additional elements

Defining additional elements is only slightly more complicated than defining additional attributes. Basically, DTD authors should write the element declaration for each element:

```
<!ELEMENT myml:myelement ( #PCDATA | myml:myotherelement )* >
<!ATTLIST myml:myelement
          myattribute    CDATA    #IMPLIED
>

<!ELEMENT myml:myotherelement EMPTY >
```

After the elements are defined, they need to be integrated into the content model. Strategies for integrating new elements or sets of elements into the content model are addressed in the next section.

6.3. Defining the content model for a collection of modules

Since the content model of XHTML modules is fully parameterized, DTD authors may modify the content model for every element in every module. The details of the DTD module interface are defined in XML DTD Modules [p.17] . However, basically there are two ways to approach this modification:

1. Re-define the "<element>.content" entity for each element.
2. Re-define one or more of the global content model entities (normally the *.extras parameter entity).

The strategy taken will depend upon the nature of the modules being combined and the nature of the elements being integrated. The remainder of this section describes techniques for integrating two different classes of modules.

6.3.1. Integrating a stand-alone module into XHTML

When a module (and remember, a module can be a collection of other modules) contains elements that only reference each other in their content model, it is said to be "internally complete". As such, the module can be used on its own (for example, you could define a DTD that was just that module, and use one of its elements as the root element). Integrating such a module into XHTML is a three step process:

1. Decide what element(s) can be thought of as the root(s) of the new module.
2. Decide where these elements need to attach in the XHTML content tree.
3. Then, for each attachment point in the content tree, add the root element(s) to the content definition for the XHTML elements.

Consider attaching the elements defined above [p.20] . In that example, the element `myelement` is the root. To attach this element under the `img` element, and only the `img` element, of XHTML, the following would work:

```
<!ENTITY % Img.content "( myml:myelement )*">
```

A DTD defined with this content model would allow a document like the following fragment:

```
<img src="...">
<myml:myelement xmlns:myml="http://www.my.org/DTDs/myml1_0.dtd">This is content of a locally defined element</myml:myelement>
</img>
```

It is important to note that normally the `img` element has a content model of `EMPTY`. By adding myelement to that content model, we are really just replacing `EMPTY` with `myml:myelement`. In the case of other elements that already have content models defined, the addition of an element would require the restating of the existing content model in addition to `myml:myelement`.

6.3.2. Mixing a new module throughout the modules in XHTML

Extending the example above, to attach this module everywhere that the `%Flow.mix` content model group is permitted, would require something like the following:

```
<!ENTITY % Misc.extra
     "| script | noscript | myml:myelement" >
```

Since the %Misc.extra content model class is used in the %Misc.class parameter entity, and that parameter entity is used throughout the XHTML Modules, the new module would become available throughout an extended XHTML document type.

6.4. Creating a new DTD

So far the examples in this section have described the methods of extending XHTML and XHTML's content model. Once this is done, the next step is to collect the modules that comprise the DTD into a single DTD driver, incorporating the new definitions so that they override and augment the basic XHTML definitions as appropriate.

When defining a new DTD, it is essential that any non-W3C elements and attributes be in their own XML Namespace. This namespace and its prefix must be declared in the document instance - either on the root element or when it is actually used.

6.4.1. Creating a simple DTD

Using the trivial example above, it is possible to define a new DTD that uses and extends the XHTML modules pretty easily. The following is a complete, working extended DTD:

```
<!ELEMENT myml:myelement ( #PCDATA | myml:myotherelement )* >
<!ATTLIST myml:myelement
     myattribute    CDATA   #IMPLIED
>

<!ELEMENT myml:myotherelement EMPTY >

<!ENTITY % Misc.extra
     "| script | noscript | myml:myelement" >

<!ENTITY % xhtml11.dtd PUBLIC "-//W3C//DTD XHTML 1.1//EN"
       "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
%xhtml11.dtd;
```

When using this DTD, it is necessary to define the XML Namespace prefix. The start of a document using this new DTD might look like:

```
<!DOCTYPE html PUBLIC "-//MYORG//DTD XHTML-MyML 1.0//EN"
                "http://www.my.org/dtd/myml1_0.dtd" >
<html xmlns="http://www.w3.org/1999/xhtml"
       xmlns:myml="http://www.my.org/dtd/myml1_0.dtd">
...
```

6.4.2. Creating a DTD by extending XHTML

Next, there is the situation where a complete, additional, and complex module is added to XHTML (or to a subset of XHTML). In essence, this is the same as in the trivial example above, the only difference being that the module being added is incorporated in the DTD by reference rather than explicitly including the new definitions in the DTD.

One such complex module is the DTD for [MathML] [p.29] . In order to combine MathML and XHTML into a single DTD, an author would just decide where MathML content should be legal in the document, and add the MathML root element to the content model at that point:

```
<!ENTITY % XHTML1-math
     PUBLIC "-//W3C//MathML 1.0//EN"
            "http://www.w3.org/DTDs/MathML/MathML1.dtd" >
%XHTML1-math;

<!ENTITY % Inlspecial.extra "a | img | object | map | mathml:math" >

<!ENTITY % xhtml11.dtd
     PUBLIC "-//W3C//XHTML 1.1//EN"
            "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd" >
%xhtml11.dtd;
```

Note that, while this is a valid example, it does not create a working DTD at this time. The reason for this is that the MathML DTD defines two elements (`var` and `select`) that conflict directly with XHTML. This conflict needs to be resolved in order for the new DTD to work correctly. Further, the elements in the MathML DTD must be declared such that they have a namespace prefix on them because XHTML requires that new elements and attributes be in their own namespaces.

6.4.3. Creating a DTD by removing and replacing XHTML modules

Another way in which DTD authors may use XHTML modules is to define a DTD that is a subset of XHTML (because, for example, they are building devices or software that only supports a subset of XHTML). Doing this is only slightly more complex than the previous example. The basic steps to follow are:

1. Take the XHTML 1.1 DTD as the basis of the new document type.
2. Select the modules to remove from that DTD.
3. Define a new DTD that "IGNOREs" the modules.

For example, consider a device that uses XHTML modules, but without forms or tables. The DTD for such a device would look like this:

```
<!ENTITY % xhtml-form.module "IGNORE" >
<!ENTITY % xhtml-table.module "IGNORE" >

<!ENTITY % xhtml11.mod
     PUBLIC "-//W3C//DTD XHTML 1.1//EN"
            "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd" >
%xhtml11.mod;
```

Note that this does not actually modify the content model for the XHTML 1.1 DTD. However, since XML ignores elements in content models that are not defined, the form and table elements are dropped from the model automatically.

6.4.4. Creating a DTD from whole cloth

Finally, some DTD authors may wish to start from scratch, using the XHTML Modularization framework as a toolkit for building a new markup language. This language must be made up of the minimal, required modules from XHTML. It may also contain other XHTML-defined modules or any other module that the author wishes to employ. In this example, we will take the basic XHTML required modules, add some XHTML-defined modules, and also add in the module we defined above.

The first step is to use the XHTML-provided template for a new module, modified for our new elements and attributes.

```
<!-- ...................................................................... -->
<!-- My Elements Module ................................................... -->
<!-- file: myelements-1_0.mod

     PUBLIC "-//MY COMPANY//ELEMENTS XHTML-MY Elements 1.0//EN"
     SYSTEM "http://www.my.org/DTDs/myelements-1_0.mod"

     xmlns:myml="http://www.my.org/DTDs/mylanguage-1_0.dtd"
     ...................................................................... -->

<!-- My Elements Module

     myns:myelement
     myns:myotherelement

     This module has no purpose other than to provide structure for some
     PCDATA content.
-->

<!ELEMENT myns:myelement ( #PCDATA | myns:myotherelement )* >
<!ATTLIST myns:myelement
        myattribute    CDATA    #IMPLIED
>

<!ELEMENT myns:myotherelement EMPTY >

<!-- end of myelements-1_0.mod -->
```

Next, use the XHTML-provided template for a new DTD, modified as appropriate for our new markup language:

```
<!-- ....................................................................... -->
<!-- MYLANGUAGE DTD  ....................................................... -->
<!-- file: mylanguage.dtd
-->

<!-- MYLANGUAGE DTD
-->
<!-- This is the DTD driver for mylanguage.

     Please use this formal public identifier to identify it:

         "-//MY COMPANY//DTD XHTML-MYML 1.0//EN"

     And this namespace for myml-unique elements:

         xmlns:myml="http://www.my.org/DTDs/mylanguage-1_0.dtd"
-->
```

```
<!ENTITY % XHTML.version  "-//MY COMPANY//DTD XHTML-MYML 1.0//EN" >

<!-- Reserved for use with the XLink namespace:
-->
<!ENTITY % XLINK.ns "" >
<!ENTITY % XLinkns.attrib "" >


<!-- reserved for future use with document profiles -->
<!ENTITY % XHTML.profile  "" >

<!-- Internationalization features
     This feature-test entity is used to declare elements
     and attributes used for internationalization support. Set it to INCLUDE
     or IGNORE as appropriate for your markup language.
-->
<!ENTITY % XHTML.I18n            "IGNORE" >

<!-- :::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::: -->

<!-- Redeclare the Misc.extra to include myelement to hook it into the
     content model.
-->

<!ENTITY % Misc.extra
     "| script | noscript | myml:myelement" >

<!-- Define the Content Model
     Remember that you can modify this content model or replace it simply be
     changing the following ENTITY declaration.
-->
<!ENTITY % xhtml-model.mod
     PUBLIC "-//W3C//ENTITIES XHTML 1.1 Document Model 1.0//EN"
     SYSTEM "http://www.w3.org/TR/xhtml11/DTD/xhtml11-model-1.mod" >

<!-- Pre-Framework Redeclaration placeholder  .................... -->
<!-- this serves as a location to insert markup declarations
     into the DTD prior to the framework declarations.
-->
<!ENTITY % xhtml-prefw-redecl.module "IGNORE" >
<![%xhtml-prefw-redecl.module;[
%xhtml-prefw-redecl.mod;
<!-- end of xhtml-prefw-redecl.module -->]]>

<!-- The events module should be included here if you need it. In this
     skeleton it is IGNOREd.
-->
<!ENTITY % xhtml-events.module "IGNORE" >

<!-- Modular Framework Module  .................................. -->
<!ENTITY % xhtml-framework.module "INCLUDE" >
<![%xhtml-framework.module;[
<!ENTITY % xhtml-framework.mod
     PUBLIC "-//W3C//ENTITIES XHTML 1.1 Modular Framework 1.0//EN"
            "xhtml11-framework-1.mod" >
%xhtml-framework.mod;]]>
```

```
<!-- Post-Framework Redeclaration placeholder  .................. -->
<!-- this serves as a location to insert markup declarations
     into the DTD following the framework declarations.
-->
<!ENTITY % xhtml-postfw-redecl.module "IGNORE" >
<![%xhtml-postfw-redecl.module;[
%xhtml-postfw-redecl.mod;
<!-- end of xhtml-postfw-redecl.module -->]]>

<!-- Basic Text Module (Required)  ............................... -->
<!ENTITY % xhtml-text.module "INCLUDE" >
<![%xhtml-text.module;[
<!ENTITY % xhtml-text.mod
     PUBLIC "-//W3C//ELEMENTS XHTML 1.1 Basic Text 1.0//EN"
            "xhtml11-text-1.mod" >
%xhtml-text.mod;]]>

<!-- Hypertext Module (required) ................................. -->
<!ENTITY % xhtml-hypertext.module "INCLUDE" >
<![%xhtml-hypertext.module;[
<!ENTITY % xhtml-hypertext.mod
     PUBLIC "-//W3C//ELEMENTS XHTML 1.1 Hypertext 1.0//EN"
            "xhtml11-hypertext-1.mod" >
%xhtml-hypertext.mod;]]>

<!-- Lists Module (required)  .................................... -->
<!ENTITY % xhtml-list.module "INCLUDE" >
<![%xhtml-list.module;[
<!ENTITY % xhtml-list.mod
     PUBLIC "-//W3C//ELEMENTS XHTML 1.1 Lists 1.0//EN"
            "xhtml11-list-1.mod" >
%xhtml-list.mod;]]>

<!-- Your modules can be included here.  Use the basic form defined above, and
     be sure to include the public FPI definition in your catalog file for
     each module that you define. You may also include W3C-defined modules at
     this point.
-->

<!-- My Elements Module   ........................................ -->
<!ENTITY % myelements.mod
     PUBLIC "-//MY COMPANY//ELEMENTS XHTML-MY Elements 1.0//EN"
            "http://www.my.org/DTDs/myelements-1_0.mod" >
%myelements.mod;>


<!-- Document Structure Module (required)  ....................... -->
<!ENTITY % xhtml-struct.module "INCLUDE" >
<![%xhtml-struct.module;[
<!ENTITY % xhtml-struct.mod
     PUBLIC "-//W3C//ELEMENTS XHTML 1.1 Document Structure 1.0//EN"
            "xhtml11-struct-1.mod" >
%xhtml-struct.mod;]]>

<!-- end of SKELETAL DTD  ......................................... -->
<!-- .......................................................... -->
```

6.5. Using the new DTD

Once a new DTD has been developed, it can be used in any document. Using the DTD is as simple as just referencing it in the DOCTYPE declaration of a document:

```
<!DOCTYPE html PUBLIC "-//MY COMPANY//DTD XHTML-MYML 1.0//EN"
          "http://www.my.org/DTDs/myorg.dtd">
<html xmlns:myml="http://www.my.org/DTDs/mylanguage-1_0.dtd">
<head>
<title>MyOrg Document</title>
</head>
<body>
<p>This is an example document using the new elements:
<myml:myelement>A test element <myml:myotherelement /> </myml:myelement>
</p>
</body>
</html>
```

A. References

This appendix is *normative*.

A.1. Normative References

[XHTML1]
XHTML 1.0: The Extensible HyperText Markup Language, Steven Pemberton, et. al., 10 December 1999.
See: http://www.w3.org/TR/xhtml1
[XML]
Extensible Markup Language (XML) 1.0: W3C Recommendation, Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, 10 February 1998.
See: http://www.w3.org/TR/REC-xml
[SGML]
Information Processing -- Text and Office Systems -- Standard Generalized Markup Language (SGML), ISO 8879:1986.
Please consult http://www.iso.ch/cate/d16387.html for information about the standard, or http://www.oasis-open.org/cover/general.html#overview about SGML.
[MATHML]
Mathematical Markup Language (MathML) 1.01 Specification: W3C Recommendation, Patrick Ion, Robert Miner, et al, 7 July 1999.
See: http://www.w3.org/TR/REC-MathML
[XMLSCHEMA]
XML Schema Part 1: Structures Henry S. Thompson, et. al., 17 December 1999
See: http://www.w3.org/TR/1999/WD-xhtmlschema-1-19991217

A.2. Informative References

[CATALOG]
Entity Management: OASIS Technical Resolution 9401:1997 (Amendment 2 to TR 9401) Paul Grosso, Chair, Entity Management Subcommittee, SGML Open, 10 September 1997.
See: http://www.oasis-open.org/html/a401.htm
[DEVDTD]
Developing SGML DTDs: From Text to Model to Markup, Eve Maler and Jeanne El Andaloussi.
Prentice Hall PTR, 1996, ISBN 0-13-309881-8.
[STRUCTXML]
Structuring XML Documents, David Megginson. Part of the Charles Goldfarb Series on Information Management.
Prentice Hall PTR, 1998, ISBN 0-13-642299-3.
[DOCBOOK]
DocBook DTD, Eve Maler and Terry Allen.
Originally created under the auspices of the Davenport Group, DocBook is now maintained by OASIS. The *Customizer's Guide for the DocBook DTD V2.4.1* is available from this site.

See: http://www.oasis-open.org/docbook/index.html

[DUBLIN]
The Dublin Core: A Simple Content Description Model for Electronic Resources, The Dublin Core Metadata Initiative.
See: http://purl.oclc.org/dc/

[SMIL]
Synchronized Multimedia Integration Language (SMIL) 1.0 Specification, Philipp Hoschka, 15 June 1998.
See: http://www.w3.org/TR/REC-smil

[TEI]
The Text Encoding Initiative (TEI)
See: http://www.uic.edu/orgs/tei/

[URI]
Uniform Resource Identifiers (URI): Generic Syntax, T. Berners-Lee, R. Fielding, L. Masinter, August 1998.
See: http://www.ietf.org/rfc/rfc2396.txt. This RFC updates RFC 1738 [URL] [p.30] and [RFC1808] [p.30] .

[URL]
IETF RFC 1738, Uniform Resource Locators (URL), T. Berners-Lee, L. Masinter, M. McCahill.
See: http://www.ietf.org/rfc/rfc1738.txt

[RFC-1808]
Relative Uniform Resource Locators, R. Fielding.
See: http://www.ietf.org/rfc/rfc1808.txt

[CSS2]
"Cascading Style Sheets, level 2 (CSS2) Specification", B. Bos, H. W. Lie, C. Lilley, I. Jacobs, 12 May 1998.
Available at: http://www.w3.org/TR/REC-CSS2

[DOM]
"Document Object Model (DOM) Level 1 Specification", Lauren Wood *et al.*, 1 October 1998.
Available at: http://www.w3.org/TR/REC-DOM-Level-1

[RFC2119]
"RFC2119: Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997.
Available at: http://www.ietf.org/rfc/rfc2119.txt

[RFC2376]
"RFC2376: XML Media Types", E. Whitehead, M. Murata, July 1998.
Available at: http://www.ietf.org/rfc/rfc2376.txt

[TIDY]
"HTML Tidy" is a tool for detecting and correcting a wide range of markup errors prevalent in HTML. It can also be used as a tool for converting existing HTML content to be well formed XML. Tidy is being made available on the same terms as other W3C sample code, i.e. free for any purpose, and entirely at your own risk.
It is available from: http://www.w3.org/Status.html#TIDY

[XMLNAMES]
"Namespaces in XML", T. Bray, D. Hollander, A. Layman, 14 January 1999.
XML namespaces provide a simple method for qualifying names used in XML documents by associating them with namespaces identified by URI.
Available at: http://www.w3.org/TR/REC-xml-names

[XMLSTYLE]
"Associating stylesheets with XML documents Version 1.0", J. Clark, 14 January 1999.
This document describes a means for a stylesheet to be associated with an XML document by including one or more processing instructions with a target of xml-stylesheet in the document's prolog.
Available at: http://www.w3.org/TR/PR-xml-stylesheet