Modularization of XHTML™

W3C Working Draft 10 September 1999

This version:
http://www.w3.org/TR/1999/WD-xhtml-modularization-19990910
(Single HTML file [p.1] , Postscript version, PDF version, ZIP archive, or Gzip'd TAR archive)
Latest version:
http://www.w3.org/TR/xhtml-modularization
Previous version:
http://www.w3.org/TR/1999/xhtml-modularization-19990406/
Editors:
Murray Altheim, Sun Microsystems
Frank Boumphrey, HTML Writers Guild
Sam Dooley, IBM
Shane McCarron, Applied Testing and Technology
Ted Wugofski, Gateway

Abstract

This working draft specifies an abstract modularization of XHTML 1.0. A companion document, Building XHTML Modules, implements this abstraction as a collection of component XML Document Type Definitions (DTDs). This modularization provide a means for subsetting and extending XHTML, a feature desired for extending XHTML's reach onto emerging platforms.

Status of this document

This document is nearly complete, and is being circulated for a final public review prior to last call.

This document is a working draft of the W3C's HTML Working Group. This working draft may be updated, replaced or rendered obsolete by other W3C 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 document is work in progress and does not imply endorsement by the W3C membership.

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)*.

Please send detailed comments on this document to www-html-editor@w3.org. We cannot guarantee a personal response, but we will try when it is appropriate. Public discussion on HTML features takes place on the mailing list www-html@w3.org.

Quick Table of Contents

Full Table of Contents

1. Introduction

This section is *normative*.

1.1. What is XHTML?

XHTML is the reformulation of HTML 4.0 as an application of XML. XHTML 1.0 [XHTML1] specifies three XML document types that correspond to the three HTML 4.0 DTDs: Strict, Transitional, and Frameset. XHTML 1.0 is the basis for a family of document types that subset and extend HTML.

1.2. What is XHTML Modularization?

XHTML Modularization is decomposition of XHTML 1.0, and by reference HTML 4.0, into a collection of abstract modules that provide specific types of functionality. These abstract modules are implemented in the XHTML 1.1 specification using the XML Document Type Definition language, but other implementations are possible and expected. The mechanism for defining the abstract modules defined in this document, and for implementing them using XML DTDs, is defined in the document "Building XHTML Modules" [BUILDING].

These modules may be combined with each other and with other modules to create XHTML subset and extension document types that qualify as members of the XHTML family of document types.

1.3. Why Modularize XHTML?

The modularization of XHTML refers to the task of specifying well-defined sets of XHTML elements that can be combined and extended by document authors, document type architects, other XML standards specifications, and application and product designers to make it economically feasible for content developers to deliver content on a greater number and diversity of platforms.

Over the last couple of years, many specialized markets have begun looking to HTML as a content language. There is a great movement toward using HTML across increasingly diverse computing platforms. Currently there is activity to move HTML onto mobile devices (hand held computers, portable phones, etc.), television devices (digital televisions, TV-based web browsers, etc.), and appliances (fixed function devices). Each of these devices has different requirements and constraints.

Modularizing XHTML provides a means for product designers to specify which elements are supported by a device using standard building blocks and standard methods for specifying which building blocks are used. These modules serve as "points of conformance" for the content community. The content community can now target the installed base that supports a certain collection of modules, rather than worry about the installed base that supports this permutation of XHTML elements or that permutation of XHTML elements. The use of standards is critical for modularized XHTML to be successful on a large scale. It is not economically feasible for content developers to tailor content to each and every permutation of XHTML elements. By specifying a standard, either software processes can autonomously tailor content to a device, or the device can automatically load the software required to process a module.

Modularization also allows for the extension of XHTML's layout and presentation capabilities, using the extensibility of XML, without breaking the XHTML standard. This development path provides a stable, useful, and implementable framework for content developers and publishers to manage the rapid pace of technological change on the Web.

The modularization of XHTML is accomplished on two major levels: at the abstract level, and at the document type level. Roughly speaking, the abstract level provides a conceptual approach to the modularization of XHTML, while the document type level provides DTD-level building blocks that allow document type designers to support the abstract modules.

1.3.1. Abstract modules

An XHTML document type is defined as a set of abstract modules. A abstract module defines, in a document type, one kind of data that is semantically different from all others. Abstract modules can be combined into document types without a deep understanding of the underlying schema that defines the modules.

1.3.2. DTD modules

A 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 a 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.

An XML DTD is a means of describing the structure of a class of XML documents, collectively known as an XML document type. XML document types are currently represented as DTDs, as described in the XML 1.0 Recommendation [XML]. Where possible, this document also allows for the potential use of other schema languages that are currently under consideration by the W3C XML Schema Working Group. (e.g. DCD, SOX, DDML, XSchema)

1.3.3. Hybrid document types

A hybrid document type is an XML DTD composed from a collection of XML DTDs or DTD Modules. The primary purpose of the modularization framework described in this document is to allow a DTD author to combine elements from multiple abstract modules into a hybrid document type, develop documents against that hybrid document type, and to validate that document against the associated hybrid document type definition.

One of the most valuable benefits of XML over SGML is that XML reduces the barrier to entry for standardization of element sets that allow communities to exchange data in an interoperable format. However, the relatively static nature of HTML as the content language for the Web has meant that any one of these communities have previously held out little hope that their XML document types would be able to see widespread adoption as part of Web standards. The modularization framework allows for the dynamic incorporation of these diverse document types within the XHTML family of document types, further reducing the barriers to the incorporation of these domain-specific vocabularies in XHTML documents.

1.3.4. Validation

The use of well-formed, but not valid, documents is an important benefit of XML. In the process of developing a document type, however, the additional leverage provided by a validating parser for error checking is important. The same statement applies to XHTML document types with elements from multiple abstract modules.

The general problem of fragment validation - validation of XML documents with different schemas from multiple XML Namespaces [XMLNAMES] [p.36] in different portions of the document - is beyond the scope of this framework. An essential feature of this framework, however, is a collection of conventions for creating, from a set of abstract modules, hybrid DTDs.

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] is highly recommended.

document type
: a class of documents sharing a common abstract structure. The ISO 8879 [SGML] [p.33] definition is as follows: "a class of documents having similar characteristics; for example, journal, article, technical manual, or memo. (4.102)"

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.

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.

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.

reference DTD
: a DTD whose markup model represents the foundation of a complete document type. A reference DTD provides the basis for the design of a "family" of related DTDs, such as subsets, extensions and variants. XHTML 1.1 [XHTML11] acts as a reference DTD for the XHTML family of document types.

subset DTD
: a DTD whose document model is the proper subset of a reference document type, whose conforming document instances are still valid according to the reference DTD. A subset may place tighter restrictions on the markup than the reference, remove elements or attributes, or both.

extension DTD
: a DTD whose document model extends a reference document type (usually by the addition of element types or attributes), but generally makes no profound changes to the reference document model other than required to add the extension’s semantic components. An extension can also be considered a proper superset if the reference document type is a proper subset of the extension.

variant DTD
: a DTD whose document model alters (through subsetting, extension, and/or substitution) the basic data model of a reference document type. It is often difficult to transform without loss between instances conforming to a variant DTD and the reference DTD. XHTML

Family Conforming Document Types are not permitted to be "variant DTDs" of the XHTML 1.1 DTD.

fragment DTD
: a portion of a DTD used as a component either for the creation of a compound or variant document type, or for validation of a document fragment. Neither SGML nor XML current have standardized methods for such partial validation.

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).

minimal content model
: Some XHTML modules define minimal content models for their elements. When these modules are used in an XHTML Family DTD, their content models cannot be altered except that they may be extended beyond that of the minimal content model defined.

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

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

element
: an instance of an element type.

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

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

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

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.,

"&amp;" for "&", "&#60;" for "<", "&copy;" for "©".)

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

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.

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.

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.

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

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.

module
: A module is a collection of elements, attributes, values for attributes, content models, or any combination of these.

3. Conformance Definition

This section is *normative*.

In order to ensure that XHTML-family documents are maximally portable among XHTML-family user agents, this specification rigidly defines conformance requirements for both of these and for XHTML-family document types. While the conformance definitions can be found in this section, they necessarily reference normative text within this document, within the base XHTML specification [XHTML1], 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. XHTML Family Document Type Conformance

It is possible to modify existing document types and define wholly new document types using both modules defined in this specification and other modules. Such a document type conforms to this specification when it meets the following criteria:

1. The document type must be defined using one of the implementation methods defined by the W3C (currently this is limited to XML DTDs, but XML Schema will be available soon).
2. The document type must have a unique identifier as defined in Naming Rules [p.14] .
3. The document type must include, at a minimum, the Structure, Hypertext, Basic Text, and List modules defined in this specification.
4. The document type must declare a unique namespace identifier that can be used as the value of the `xmlns` attribute and that is defined as the `FIXED` value of the `xmlns` attribute of the `html` element defined in the Structure Module. When this identifier is expressed as a URI, the URI must dereference to the implementation of the document type.
5. For each of the W3C-defined modules that are included, all of the elements, attributes, and any required minimal content models must be included (and optionally extended) in the document type's content model.
6. The document type may define additional elements. However, these elements must not have the same name as any other W3C-defined elements.

3.2. XHTML Family Document Conformance

Documents that rely upon XHTML-family document types are considered XHTML conforming if they validate against their referenced document type.

3.3. XHTML Family User Agent Conformance

A conforming user agent must meet all of the following criteria (as defined in [XHTML1]):

1. In order to be consistent with the XML 1.0 Recommendation [XML] [p.??] , the user agent must parse and evaluate an XHTML document for well-formedness. If the user agent claims to be a validating user agent, it must also validate documents against their referenced DTDs according to [XML].
2. When the user agent claims to support facilities defined within this specification or required by this specification through normative reference, it must do so in ways consistent with the facilities' definition.
3. When a user agent processes a document of Internet media type `text/xml`, it shall only recognize attributes of type `ID` (e.g. the `id` attribute on most XHTML elements) as fragment identifiers.
4. If a user agent encounters an element it does not recognize, it must render the element's content.
5. If a user agent encounters an attribute it does not recognize, it must ignore the entire attribute specification (i.e., the attribute and its value).
6. If a user agent encounters an attribute value it doesn't recognize, it must use the default attribute value.
7. If it encounters an entity reference (other than one of the predefined entities) for which the User Agent has processed no declaration (which could happen if the declaration is in the external subset which the User Agent hasn't read), the entity reference should be rendered as the characters (starting with the ampersand and ending with the semi-colon) that make up the entity reference.
8. When rendering content, User Agents that encounter characters or character entity references that are recognized but not renderable should display the document in such a way that it is obvious to the user that normal rendering has not taken place.
9. XML does not specifically define whitespace handling characteristics for elements where the `xml:space` attribute is set to `default`. For all such elements, XHTML User Agents are required to suppress line breaks occurring immediately after the start tag or immediately prior to the end tag.

3.4. 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 document types implemented as XML Document Type Definitions 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 `DTD XHTML-` followed by an organization-defined

unique identifier (e.g. MyML 1.0). This identifier 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 family conforming document type might be `-//MyCompany//DTD XHTML-MyML 1.0//EN`.

3.4.1. 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 iwth a specific version of XHTML - the name does not need to identify version dependencies. In the case of new document types, the new type does not necessarily have any relationship to a specific version of XHTML. Instead, the new document type should itself have versioning that will help iun its evolution. Document types will necessarily evolve out of step with XHTML from the W3C.

4. XHTML Abstract Modules

This section is *normative*.

This section specifies the contents of the XHTML abstract modules. These modules are abstract definitions of collections of elements, attributes, and their content models. These abstract modules can be mapped onto any appropriate specification mechanism. The XHTML 1.1 Specification, for example, maps these modules onto DTDs as described in [XML].

Content developers and device designers should view this section as a guide to the definition of the functionality provided by the various XHTML-defined modules. When developing documents or defining a profile for a class of documents, content developers can determine which of these modules are essential for conveying their message. When designing clients, device designers should develop their device profiles by choosing from among the abstract modules defined here.

4.1. Common Characteristics of Modules

Many of the abstract modules in this section describe elements, attributes on those elements, and minimal content models for those elements or element sets. This section identifies some shorthand expressions that are used throughout the abstract module definitions. These expressions should in no way be considered normative or mandatory. They are an editorial convenience for this document. When used in the remainder of this section, it is the expansion of the term that is normative, not the term itself.

4.1.1. Syntactic Conventions

The abstract modules are not defined in a formal grammar. However, the definitions do adhere to the following syntactic conventions (as defined in Building XHTML Modules [BUILDING [p.33]]). 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.

4.1.2. Content Types

The abstract module definitions in this document 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.

4.1.3. Attribute Types

In some instances, the types of attribute values or the explicit set of permitted values for attributes are defined. 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

4.1.4. Attribute Collections

The following basic attribute sets are used on many elements. In each case where they are used, their use is identified via their name rather than enumerating the list.

Collection Name	Attributes in Collection
Core	class (NMTOKEN), id (ID), title (CDATA)
I18N	dir ("rtl" | "ltr"), xml:lang (NMTOKEN)
Events	onclick, ondblclick, onmousedown, onmouseup, onmouseover, onmousemove, onmouseout, onkeypress, onkeydown, onkeyup
Style	style (CDATA)
Common	Core + Events + Internationalization + Style

Note that the Events collection is only defined when the Intrinsic Events abstract module is selected. Otherwise, the Events collection is empty.

Also note that the Style collection is only defined when the Stylesheet Module is selected. Otherwise, the Style collection is empty.

4.2. Basic Modules

The basic modules are modules that are required to be present in any XHTML Family Conforming Document Type [p.13] .

4.2.1. Structure Module

The Structure Module defines the major structural elements for XHTML. These elements effectively act as the basis for the content model of many XHTML family document types. The elements and attributes included in this module are:

Elements	Attributes	Minimal Content Model
body	Common	(Heading | Block | List)*
div	Common	(Heading | Block | List)*
head	I18n, profile	title
html	I18n, version, xmlns	head, body
span	Common	(PCDATA | Inline)*
title	I18n	PCDATA

This module is the basic structural definition for XHTML content. The `html` element acts as the root element for all XHTML Family Document Types. The `div` element is added to the Block content set and the `span` element is added to the Inline content set as these are defined in the Basic Text Module below.

4.2.2. Basic Text Module

This module defines all of the basic text container elements, attributes, and their content model:

Element	Attributes	Minimal Content Model
abbr	Common	(PCDATA | Inline)*
acronym	Common	(PCDATA | Inline)*
address	Common	(PCDATA | Inline)*
blockquote	Common, cite	(PCDATA | Heading | Block)*
br	Core	EMPTY
cite	Common	(PCDATA | Inline)*
code	Common	(PCDATA | Inline)*
dfn	Common	(PCDATA | Inline)*
em	Common	(PCDATA | Inline)*
h1	Common	(PCDATA | Inline)*
h2	Common	(PCDATA | Inline)*
h3	Common	(PCDATA | Inline)*
h4	Common	(PCDATA | Inline)*
h5	Common	(PCDATA | Inline)*
h6	Common	(PCDATA | Inline)*
kbd	Common	(PCDATA | Inline)*
p	Common	(PCDATA | Inline)*
pre	Common	(PCDATA | Inline)*
q	Common	(PCDATA | Inline)*
samp	Common	(PCDATA | Inline)*
strong	Common	(PCDATA | Inline)*
var	Common	(PCDATA | Inline)*

The minimal content model for this module defines some content sets:

Heading
: h1 | h2 | h3 | h4 | h5 | h6

Block
: address | blockquote | p | pre

Inline
    abbr | acronym | br | cite | code | dfn | em | kbd | q | samp | strong | var
Flow
    Heading | Block | Inline

4.2.3. Hypertext Module

The Hypertext Module provides the element that is used to define hypertext links to other resources. This module supports the following element and attributes:

Element	Attributes	Minimal Content Model
a	Common, charset, href, hreflang, rel, rev, type	(PCDATA | Inline - a)*

This module adds the a element to the Inline content set of the Basic Text Module.

4.2.4. List Module

As its name suggests, the List Module provides list-oriented elements. Specifically, the List Module supports the following elements and attributes:

Elements	Attributes	Minimal Content Model
dl	Common	(dt | dd)+
dt	Common	(PCDATA | Inline)*
dd	Common	(PCDATA | Inline)*
ol	Common	li+
ul	Common	li+
li	Common	(PCDATA | Inline)*

This module also defines the content set List with the minimal content model (dl | ol | ul)+ and adds this set to the Flow content set of the Basic Text Module.

4.3. Applet Module

The Applet Module provides elements for referencing external applications. Specifically, the Applet Module supports the following elements and attributes:

Element	Attributes	Minimal Content Model
applet	Core, alt, archive, code, codebase, height, name, object, width	param?
param	id (ID), name (CDATA), type, value, valuetype	EMPTY

When the Applet Module is used, it adds the `applet` element to the Inline content set of the Basic Text Module.

4.4. Text Extension Modules

This section defines a variety of additional textual markup modules.

4.4.1. Presentation Module

This module defines elements, attributes, and a minimal content model for simple presentation-related markup:

Element	Attributes	Minimal Content Model
b	Common	(PCDATA | Inline)*
big	Common	(PCDATA | Inline)*
hr	Common	EMPTY
i	Common	(PCDATA | Inline)*
small	Common	(PCDATA | Inline)*
sub	Common	(PCDATA | Inline)*
sup	Common	(PCDATA | Inline)*
tt	Common	(PCDATA | Inline)*

When this module is used, the `hr` element is added to the Block content set of the Basic Text Module. In additional, the `b, big, i, small, sub, sup,` and `tt` elements are added to the Inline content set of the Basic Text Module.

4.4.2. Edit Module

This module defines elements and attributes for use in editing-related markup:

Element	Attributes	Minimal Content Model
del	Common	(PCDATA | Inline)*
ins	Common	(PCDATA | Inline)*

When this module is used, the `del` and `ins` elements are added to the Inline content set of the Basic Text Module.

4.4.3. BDO Module

The BDO module defines an element that can be used to declare the bi-directional rules for the element’s content.

Elements	Attributes	Minimal Content Model
bdo	Common	(PCDATA | Inline)*

When this module is used, the `bdo` element are added to the Inline content set of the Basic Text Module.

4.5. Forms Modules

4.5.1. Basic Forms Module

The Basic Forms Module provides the forms features found in HTML 3.2. Specifically, the Basic Forms Module supports the following elements, attributes, and minimal content model:

Elements	Attributes	Minimal Content Model
form	Common, action, method, enctype	Heading | Block - form
input	Common, checked, maxlength, name, size, src, type, value	EMPTY
select	Common, multiple, name, size	option+
option	Common, selected, value	Inline*
textarea	Common, columns, name, rows	PCDATA*

This module defines two content sets:

Form
    form

Formctrl
    input | select | textarea

When this module is used, it adds the Form content set to the Block content set and it adds the Formctrl content set to the Inline content set as these are defined in the Basic Text Module.

4.5.2. Forms Module

The Forms Module provides all of the forms features found in HTML 4.0. Specifically, the Forms Module supports:

Elements	Attributes	Minimal Content Model
form	Common, accept, accept-charset, action, method, enctype	(Heading | Block - form | fieldset)+
input	Common, accept, accesskey, alt, checked, disabled, maxlength, name, readonly, size, src, tabindex, type, value	EMPTY
select	Common, disabled, multiple, name, size, tabindex	(optgroup | option)+
option	Common, disabled, label, selected, value	PCDATA
textarea	Common, accesskey, columns, disabled, name, readonly, rows, tabindex	PCDATA
button	Common, accesskey, disabled, name, tabindex, type, value	(PCDATA | Heading | List | Block - Form | Inline - Formctrl)*
fieldset	Common	(PCDATA | legend | Flow)*
label	Common, accesskey, for	(PCDATA | Inline - label)*
legend	Common, accesskey	(PCDATA | Inline)+
optgroup	Common, disabled, label	option+

This module defines two content sets:

Form
    form | fieldset
Formctrl
    input | select | textarea | label | button

When this module is used, it adds the Form content set to the Block content set and it adds the Formctrl content set to the Inline content set as these are defined in the Basic Text Module.

The Forms Module is a superset of the Basic Forms Module. These modules may not be used together in a single document type.

4.6. Table Modules

4.6.1. Basic Tables Module

The Basic Tables Module provides table-related elements, but only in a limited form. Specifically, the Basic Tables Module supports:

Elements	**Attributes**	**Minimal Content Model**
caption	Common	(PCDATA | Inline)*
table	Common, border, cellpadding. cellspacing, summary, width	caption?, tr+
td	Common, abbr, align, axis, colspan, headers, rowspan, scope, valign	(PCDATA | Flow)*
th	Common, abbr, align, axis, colspan, headers, rowspan, scope, valign	(PCDATA | Flow)*
tr	Common, align, valign	(th | td)+

When this module is used, it adds the `table` element to the Block content set as defined in the Basic Text Module.

4.6.2. Tables Module

As its name suggests, the Tables Module provides table-related elements that are better able to be accessed by non-visual user agents. Specifically, the Tables Module supports the following elements, attributes, and content model:

Elements	Attributes	Minimal Content Model
caption	Common	(PCDATA | Inline)*
table	Common, border, cellpadding. cellspacing, datapagesize, frame, rules, summary, width	caption?, (col* | colgroup*), ((thead?, tfoot?, tbody+) | (tr+))
td	Common, abbr, align, axis, colspan, headers, rowspan, scope, valign	(PCDATA | Inline)*
th	Common, abbr, align, axis, colspan, headers, rowspan, scope, valign	(PCDATA | Inline)*
tr	Common, align, valign	(td | th)+
col	Common, align, span, valign, width	EMPTY
colgroup	Common, align, span, valign, width	col*
tbody	Common, align, valign	tr+
thead	Common, align, valign	tr+
tfoot	Common, align, valign	tr+

When this module is used, it adds the `table` element to the Block content set of the Basic Text Module.

4.7. Image Module

The Image Module provides basic image embedding, and may be used in some implementations independently of client side image maps. The Image Module supports the following element and attributes:

Elements	Attributes	Minimal Content Model
img	Common, alt, height, longdesc, src, width	EMPTY

When this module is used, it adds the `img` element to the Inline content set of the Basic Text Module.

4.8. Client-side Image Map Module

The Client-side Image Map Module provides elements for client side image maps. It requires that the Image Module (or another module that supports the `img` element) be included. The Client-side Image Map Module supports the following elements:

Elements	Attributes	Minimal Content Model
a&	coords, shape	n/a
area	Common, accesskey, alt, coords, href, nohref, shape, tabindex	EMPTY
img&	usemap	n/a
map	Common, name	((Heading | Block) | area)+
object&	usemap	Note: Only when the object module is included

When this module is used, the `table` element is added to the Block content set of the Basic Text Module.

4.9. Server-side Image Map Module

The Server-side Image Map Module provides support for image-selection and transmission of selection coordinates. It requires that the Image Module (or another module that supports the `img` element) be included. The Server-side Image Map Module supports the following attributes:

Elements	Attributes	Minimal Content Model
img&	ismap	n/a

4.10. Object Module

The Object Module provides elements for general-purpose object inclusion. Specifically, the Object Module supports:

Elements	Attributes	Minimal Content Model
object	Common, archive, classid, codebase, codetype, data, declare, height, standby, tabindex, type, width	(PCDATA | Flow | param)*
param	id, type, value, valuetype	EMPTY

When this module is used, it adds the `object` element to the Inline content set of the Basic Text Module.

4.11. Frames Module

As its name suggests, the Frames Module provides frame-related elements. Specifically, the Frames Module supports:

Elements	Attributes	Minimal Content Model
frameset	Core, cols, rows	(frame | noframes)+
frame	Core, frameborder, longdesc, marginheight, marginwidth, noresize, scrolling, src	EMPTY
noframes	Common	body
a&	target	n/a

4.12. Iframe Module

The Iframe Module defines an element that can be used to define a base URL against which relative URIs in the document will be resolved. The element and attribute included in this module are:

Elements	Attributes	Minimal Content Model
iframe	Core, frameborder, height, longdesc, marginheight, marginwidth, scrolling, src, width	Flow

When this module is used, the `iframe` element is added to the Block content set as defined by the Basic Text Module.

4.13. Intrinsic Events

Intrinsic events are attributes that are used in conjunction with elements that can have specific actions occur when certain events are performed by the user. The attributes indicated in the following table are added to the attribute set for their respective elements ONLY when the modules defining those elements are selected. Note also that selection of this module defines the attribute collection Events [p.19] as described above. Attributes defined by this module are:

Elements	Attributes	Notes
a&	onblur, onfocus	
area&	onblur, onfocus	When the Client-side Image Map module is also used
form&	onreset, onsubmit	When the Basic Forms or Forms module is used
body&	onload, onunload	
label&	onblur, onfocus	When the Forms module is used
input&	onblur, onchange, onfocus, onselect	When the Basic Forms or Forms module is used
select&	onblur, onchange, onfocus	When the Basic Forms or Forms module is used
textarea&	onblur, onchange, onfocus, onselect	When the Basic Forms or Forms module is used
button&	onblur, onfocus	When the Forms module is used

4.14. Metainformation Module

The Metainformation Module defines an element that describes information within the declarative portion of a document (in XHTML within the head element). This module includes the following element:

Elements	Attributes	Minimal Content Model
meta	I18n, content, http-equiv, name, scheme	EMPTY

4.15. Scripting Module

The Scripting Module defines elements that are used to contain information pertaining to executable scripts or the lack of support for executable scripts. Elements and attributes included in this module are:

Elements	Attributes	Minimal Content Model
noscript	Common	(Heading | List | Block)+
script	charset, defer, src, type	PCDATA

When this module is used, it adds the `script and noscript` elements are added to the Block content set of the Basic Text Module.

4.16. Stylesheet Module

The Stylesheet Module enables style sheet processing. Note also that selection of this module defines the attribute collection Style [p.19] as described above. The element and attributes defined by this module are:

Elements	Attributes	Minimal Content Model
style	I18n, media, title, type	PCDATA

When this module is used, it adds the `style` element to the Block content set of the Basic Text Module.

4.17. Link Module

The Link Module defines an element that can be used to define links to external resources. These resources are often used to augment the user agent’s ability to process the associated XHTML document. The element and attributes included in this module are:

Elements	Attributes	Minimal Content Model
link	Common, charset, href, hreflang, media, rel, rev, type	EMPTY

When this module is used, it adds the `link` element to the content model of the `head` element as defined in the Structure Module.

4.18. Base Module

The Base Module defines an element that can be used to define a base URL against which relative URIs in the document will be resolved. The element and attribute included in this module are:

Elements	Attributes	Minimal Content Model
base	href	EMPTY

When this module is used, it adds the `base` element to the content model of the `head` element of the Structure Module.

A. References

This appendix is *normative*.

A.1. Normative References

[BUILDING]
Building XHTML Modules: W3C Working Draft, Murray Altheim, Shane P. McCarron, 9 September 1999.
See: http://www.w3.org/TR/1999/WD-xhtml-building-19990910

[HTML40]
HTML 4.0 Specification: W3C Recommendation, Dave Raggett, Arnaud Le Hors, Ian Jacobs, 24 April 1998.
See: http://www.w3.org/TR/REC-html40

[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.

[XHTML1]
XHTML 1.0: The Extensible HyperText Markup Language, Steven Pemberton, et. al., 24 August 1999.
See: http://www.w3.org/TR/xhtml1

[XHTML11]
XHTML 1.1: Module-based XHTML, Murray Altheim, Shane McCarron, 16 August 1999.
See: http://www.w3.org/TR/1999/WD-xhtml11-199900910

[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

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.

[SGML-XML]
Comparison of SGML and XML: W3C Note, James Clark, 15 December 1997.
See: http://www.w3.org/TR/NOTE-sgml-xml-971215

[XLINK]
XML Linking Language (XLink): W3C Working Draft, Eve Maler and Steve DeRose, 3 March 1998.
A new XLink requirements document is expected soon, followed by a working draft update.
See: http://www.w3.org/TR/WD-xlink

[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/

[HTML32]
HTML 3.2 Reference Specification: W3C Recommendation, Dave Raggett, 14 January 1997.
See: http://www.w3.org/TR/REC-html32

[ISO-HTML]
ISO/IEC 15445:1998 HyperText Markup Language (HTML), David M. Abrahamson and Roger Price.
See: http://dmsl.cs.uml.edu/15445/FinalCD.html

[RDF]
Resource Description Framework (RDF): Model and Syntax Specification, Ora Lassila and Ralph R. Swick, 19 August 1998.
See: http://www.w3.org/TR/PR-rdf-syntax

[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.??] and [RFC1808] [p.??] .

[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
[CC/PP]
"Composite Capability/Preference Profiles (CC/PP): A user side framework for content negotiation", F. Reynolds, J. Hjelm, S. Dawkins, S. Singhal, 30 November 1998.
This document describes a method for using the Resource Description Format (RDF) to create a general, yet extensible framework for describing user preferences and device capabilities. Servers can exploit this to customize the service or content provided.
Available at: http://www.w3.org/TR/NOTE-CCPP
[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
[ERRATA]
"HTML 4.0 Specification Errata".
This document lists the errata for the HTML 4.0 specification.
Available at: http://www.w3.org/MarkUp/html40-updates/REC-html40-19980424-errata.html
[HTML]
"HTML 4.0 Specification", D. Raggett, A. Le Hors, I. Jacobs, 18 December 1997, revised 24 April 1998.
Available at: http://www.w3.org/TR/REC-html40
[POSIX.1]
"ISO/IEC 9945-1:1990 Information Technology - Portable Operating System Interface (POSIX) - Part 1: System Application Program Interface (API) [C Language]", Institute of Electrical and Electronics Engineers, Inc, 1990.
[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
[RFC2396]
"RFC2396: Uniform Resource Identifiers (URI): Generic Syntax", T. Berners-Lee, L. Masinter, August 1998.
This document updates RFC1738 and RFC1808.
Available at: http://www.ietf.org/rfc/rfc2396.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

[FRMWRK]
"Protocol-independent content negotiation framework", Klyne G., 16 February 1999.
See: http://www.ietf.org/internet-drafts/draft-ietf-conneg-requirements-02.txt

[SYNTAX]
"A syntax for describing media feature sets", Klyne G., 14 December 1998.
See http://www.ietf.org/internet-drafts/draft-ietf-conneg-feature-syntax-04.txt

[XMLMOD]
"XML Modularization of HTML 4.0", M. Altheim, Sun Microsystems, 2 February 1999
See http://www.altheim.com/specs/xhtml/NOTE-xhtml-modular.html

[REC FRAG]
XML Fragment Interchange - Working Draft Paul Grosso, et. al., 3 March, 1999
See: http://www.w3.org/TR/WD-xml-fragment

[TC9601]
SGML standard Technical Corrigendum 9601 Paul Grosso??
See:

[MSIE5]
Microsoft Internet Explorer Version 5.0
See: http://www.microsoft.com/windows/ie/ie5/default.asp

[XSCHEMA]
XML Schema Requirements - W3C Note Ashok Malhotra, et. al., 15 February 1999
See: http://www.w3.org/TR/1999/NOTE-xml-schema-req-19990215

B. Design Goals

This appendix is **informative**

There are six major design goals for the modularization framework for XHTML:

- [G1] Provide a means for the W3C and third parties to integrate XHTML into other XML languages.
- [G2] Provide a means for the W3C to extend XHTML with new or optional features.
- [G3] Provide a means for third parties to extend XHTML with domain-specific features.
- [G4] Provide a means for third parties to integrate other XML languages into XHTML.
- [G5] Improve the ability to create a close approximation to the HTML 4.0 DTDs.
- [G6] Improve ease-of-use for DTD developers.

B.1. Requirements

The design goals listed in the previous section lead to a large number of requirements for the modularization framework. These requirements, summarized in this section, can be further classified according to the major features of the framework to be described.

B.1.1. Granularity

Collectively the requirements in this section express the desire that the modules defined within the framework hit the right level of granularity:

- [R1.1] Abstract modules should promote and maintain content portability.
- [R1.2] Abstract modules should promote platform profile standardization.
- [R1.3] Abstract modules should be large enough to promote interoperability.
- [R1.4] Abstract modules should be small enough to avoid the need for subsets.
- [R1.5] Abstract modules should collect elements with similar or related semantics.
- [R1.6] Abstract modules should separate elements with dissimilar or unrelated semantics.
- [R1.7] Modules should be small enough to allow single element document type modules.

B.1.2. Composibility

The composibility requirements listed here are intended to ensure that the modularization framework be able to express the right set of target modules required by the communities that will be served by the framework:

- [R2.1] The module framework should allow construction of abstract modules for XHTML 1.0.
- [R2.2] The module framework should allow construction of abstract modules that closely approximate HTML 4.0.
- [R2.3] The module framework should allow construction of abstract modules for other W3C Recommendations.
- [R2.4] The module framework should allow construction of abstract modules for other XML

document types.
- [R2.5] The module framework should allow construction of abstract modules for a wide range of platform profiles.

B.1.3. Ease of Use

The modularization framework will only receive widespread adoption if it describes mechanisms that make it easy for our target audience to use the framework:

- [R3.1] The module framework should make it easy for document type designers to subset and extend XHTML abstract modules.
- [R3.2] The module framework should make it easy for document type designers to create abstract modules for other XML document types.
- [R3.3] The module framework should make it easy for document authors to validate elements from different abstract modules.

B.1.4. Compatibility

The intent of this document is that the modularization framework described here should work well with the XML and other standards being developed by the W3C Working Groups:

- [R4.1] The module framework should strictly conform to the XML 1.0 Recommendation.
- [R4.2] The module framework should be compatible with the XML linking specification.
- [R4.3] The module framework should be compatible with the XML stylesheet specification.
- [R4.4] The module framework should be able to adopt new W3C recommendations where appropriate.
- [R4.5] The module framework should not depend on W3C work in progress.
- [R4.6] The module framework should not depend on work done outside W3C.

B.1.5. Conformance

The effectiveness of the framework will also be measured by how easy it is to test the behavior of modules developed according to the framework, and to test the documents that employ those modules for validation:

- [R5.1] It should be possible to validate documents constructed using elements and attributes from abstract modules.
- [R5.2] It should be possible to explicitly describe the behavior of elements and attributes from abstract modules.
- [R5.3] It should be possible to verify the behavior of elements and attributes from abstract modules.
- [R5.4] It should be possible to verify a compound document type as an XHTML document type.
- [R5.5] Modules defined in accordance with the methods in this document shall not duplicate the names of elements or parameter entities defined in XHTML modules.