Wireless Markup Language

Version 2.0

Version 11-Sep-2001

Wireless Application Protocol
WAP-238-WML-20010911-a

A list of errata and updates to this document is available from the WAP Forum™ Web site, http://www.wapforum.org/, in the form of SIN documents, which are subject to revision or removal without notice.

This document is available online in PDF format at http://www.wapforum.org/.

Known problems associated with this document are published at http://www.wapforum.org/.

Comments regarding this document can be submitted to the WAP Forum™ in the manner published at http://www.wapforum.org/.

Document History	
WAP-238-WML-20010911-a	Current
WAP-238_100-WML-20010911-a	SIN

Contents

1. Scope

Wireless Application Protocol (WAP) is a result of continuous work to define an industry-wide specification for developing applications that operate over wireless communication networks. The scope for the WAP Forum is to define a set of specifications to be used by service applications. The wireless market is growing very quickly and reaching new customers and services. To enable operators and manufacturers to meet the challenges in advanced services, differentiation, and fast/flexible service creation, WAP defines a set of protocols in transport, session, and application layers. For additional information on the WAP architecture, refer to [WAPARCH].

This specification defines the Wireless Markup Language version 2.0 (WML2), a language that extends XHTML with the unique semantics of WML1 [WML1].

WML2 is used for backwards compatibility only. It is not intended for content authoring. WAP2 content is created with XHTML Mobile Profile [XHTMLMP].

The complete behaviour for a WAE user agent is not described in this specification, only the WML2 specific behaviour. In order to build a WAE compliant user agent, this specification must be read in conjunction with the Wireless Application Environment Specification [WAE].

2. References

2.1. Normative References

[CREQ] "Specification of WAP Conformance Requirements", WAP Forum™, WAP-221-CREQ-20000915-a. URL: http://www.wapforum.org/

[HTML4] "HTML 4.01 Specification", W3C Recommendation 24 December 1999. URL: http://www.w3.org/TR/1999/REC-html401-19991224, D. Raggett et al., December 1999.

[RFC2045] "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", N. Freed, et al., November 1996. URL: http://www.ietf.org/rfc/rfc2045.txt

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

[RFC2616] "Hypertext Transfer Protocol - HTTP/1.1", R. Fielding et al., June 1999. URL: http://www.ietf.org/rfc/rfc2616.txt

[RFC2388] "Returning Values from Forms: multipart/form-data" L. Masinter, August 1998. URL: http://www.ietf.org/rfc/rfc2388.txt

[RFC2396] "Uniform Resource Identifiers (URI): Generic Syntax", T. Berners-Lee et al., August 1998. URL: http://www.ietf.org/rfc/rfc2396.txt

[RFC3066] "Tags for the Identification of Languages", H. Alvestrand, January 2001. URL: http://www.ietf.org/rfc/rfc3066.txt

[UNICODE] "The Unicode Standard: Version 2.0", The Unicode Consortium, Addison-Wesley Developers Press, 1996. URL: http://www.unicode.org/

[WCSS] "WAP CSS Specification", WAP Forum™, WAP-239-WCSS. URL:http//www.wapforum.org/

[XHTMLBasic] "XHTML™ Basic 1.0", W3C Recommendation 19 December 2000, M. Ishikawa et al. URL: http://www.w3.org/TR/2000/REC-xhtml-basic-20001219

[XHTMLMod] "Modularization of XHTML™", W3C Recommendation 10 April 2001, M. Altheim et al. URL: http://www.w3.org/TR/2001/REC-xhtml-modularization-20010410

[XHTMLMP] "XHTML Mobile Profile", WAP Forum™. WAP-277-XHTMLMP. URL:http://www.wapforum.org/

[XML] "Extensible Markup Language (XML) 1.0 (Second Edition) ", W3C Recommendation 6 October 2000, T. Bray et al. URL: http://www.w3.org/TR/2000/REC-xml-20001006

[XMLN] "Namespaces in XML", 14 January 1999, W3C Recommendation 14 January 1999, T. Bray et al. URL: http://www.w3.org/TR/1999/REC-xml-names-19990114

2.2. Informative References

[CACHE] "WAP Caching Model Specification", WAP Forum™, WAP-120-CachingMod-19990211-a. URL: http://www.wapforum.org/

[CSSMP] "CSS Mobile Profile 1.0", W3C Working Draft, 29 January 2001, T. Wugofski et al. URL: http://www.w3.org/TR/2001/WD-css-mobile-20010129

[CSS2] "Cascading Style Sheets, level 2", B. Bos et. al., 12 May 1998.

URL: http://www.w3.org/TR/1998/REC-CSS2-19980512.

[DOM2EVENT] "Document Object Model (DOM) Level 2 Events Specification Version 1", W3C Recommendation 13 November 2000, T. Pixley.
URL: http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113

[ISO8879] "Information Processing - Text and Office Systems - Standard Generalised Markup Language (SGML) ", ISO 8879:1986.

[PICTO] "WAP Pictogram", WAP Forum™, WAP-213-WAPInterPic.
URL: http://www.wapforum.org/

[WAE] "Wireless Application Environment Specification", WAP Forum™, WAP-236-WAE.
URL: http://www.wapforum.org/

[WAPARCH] "WAP Architecture", WAP Forum™, WAP-210-WAPArch-20010712-a.
URL: http://www.wapforum.org/URL:http//www.wapforum.org/

[WML1] "Wireless Markup Language Version 1.3", WAP Forum™, WAP-191-WML-20000219-a.
URL: http://www.wapforum.org/

[WMLScript] "WMLScript Language Specification" , WAP Forum™, WAP-193-WMLScript-20000324-a.
URL: http://www.wapforum.org/

[WTA] "Wireless Telephony Application Specification ", WAP Forum™, WAP-266-WTA.
URL: http://www.wapforum.org/

[XHTML] "XHTML™ 1.1 - Module-based XHTML", W3C Recommendation 31 May 2001, S. Pemberton et al.
URL: http://www.w3.org/TR/2001/REC-xhtml11-20010531

3. Terminology and Conventions

3.1. Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

All sections and appendixes, except “Scope” and “Introduction”, are normative, unless they are explicitly indicated to be informative.

3.2. Definitions

Author - an author is a person or program that writes or generates WML, or other content.

Card - a single WML unit of navigation and user interface. May contain information to present to the user, instructions for gathering user input, etc.

Character Encoding – when used as a verb, character encoding refers to conversion between sequence of characters and a sequence of bytes. When used as a noun, character encoding refers to a method for converting a sequence of bytes to a sequence of characters. Typically, WML document character encoding is captured in transport headers attributes (e.g., Content-Type’s “charset” parameter) or the XML declaration defined by [XML].

Client - a device (or application) that initiates a request for connection with a server.

Content - subject matter (data) stored or generated at an origin server. Content is typically displayed or interpreted by a user agent in response to a user request.

Device - a network entity that is capable of sending and receiving packets of information and has a unique device address. A device can act as both a client and a server within a given context or across multiple contexts. For example, a device can service a number of clients (as a server) while being a client to another server.

Man-Machine Interface - a synonym for user interface.

Resource - a network data object or service that can be identified by a URL. Resources may be available in multiple representations (e.g., multiple languages, data formats, size and resolutions) or vary in other ways.

Server - a device (or application) that passively waits for connection requests from one or more clients. Once a connection is established, it is used to deliver a Resource to the client.

SGML - the Standard Generalised Markup Language (defined in [ISO8879]) is a general-purpose language for domain-specific markup languages.

User - a user is a person who interacts with a user agent to view, hear or otherwise use a resource.

User Agent - a user agent is any software or device that interprets WML, WMLScript, WTAI or other resources. This may include textual browsers, voice browsers, search engines, etc.

XML - the Extensible Markup Language is a World Wide Web Consortium (W3C) standard for Internet markup languages, of which WML is one such language. XML is a restricted subset of SGML.

3.3. Abbreviations

CSS	Cascading Style Sheet
DTD	Document Type Definition
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDL	Interface Definition Language
RFC	Request For Comments
SGML	Standard Generalised Markup Language
UI	User Interface
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WAE	Wireless Application Environment
WAP	Wireless Application Protocol
WML	Wireless Markup Language

WSP	Wireless Session Protocol
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language

4. Introduction (Informative)

This specification describes the processing of the WML2 document type by the WAE user agent. This section describes the WML2 document type, and its relationship to the XHTML Mobile Profile [XHTMLMP] document type.

4.1. Background

The primary requirements for the WAP2 service/application authoring environment are twofold:

- Backward compatibility with WAP 1 services
- Convergence with existing and evolving Internet standards, in particular the W3C specifications XHTML Basic and CSS Mobile Profile

The goal of WML2 is therefore to create a document type that extends the syntax and semantics of XHTML Basic [XHTMLBasic] and CSS Mobile Profile (convergence) with the unique semantics of WML1 (backward compatibility).

The design guidelines for WML2 were:

- Defer to XHTML in the case of duplicated semantics (elements, attributes, and attribute values).
- Remove WML elements, attributes, and attribute values when they can be expressed in XHTML and CSS.
- Include WML1 elements and attributes when WML1 features cannot be expressed in XHTML and CSS. These elements are included using the WML namespace, identified by the “wml:” prefix.
- Deprecate elements and attributes where the W3C has also deprecated them.

The result is a document type with an XHTML core and WAP extensions. The core document type, known as XHTML Mobile Profile, can be used to author content convergent with W3C specifications. The WML2 document type (XHTML core plus WAP extensions) can be used to deliver WML1 content to WAP 2 clients, achieving backward compatibility with WML1 in a manner that is transparent to the end user. The structure and relationship of the document types allows an efficient implementation of a user agent that supports both types.

4.2. The XHTML Mobile Profile Document Type

The XHTML Mobile Profile document type is a strict superset of the XHTML Basic document type. XHTML Mobile Profile is used as an authoring language for WAP 2 services/applications. This document type achieves convergence with XHTML for WAP 2 services/applications.

The XHTML Mobile Profile document type is defined by [XHTMLMP].

The WML2 document type extends the XHTML Mobile Profile document type. The WML2 document type is detailed in section 4.3.

4.3. The WML2 Document Type

The WML2 document type is a strict superset of the XHTML Mobile Profile document type. It adds extensions to XHTML Mobile Profile for WML1 compatibility, creating a document type that is used for achieving backward compatibility with services/applications written using WML1.

The WML2 document type defines the following elements and attributes.

In particular, it includes the WAP extension elements, those elements brought from WML1 because the equivalent capabilities are not found in XHTML:

Element

wml:access	wml:onevent
wml:anchor	wml:postfield
wml:card	wml:prev
wml:do	wml:refresh
wml:getvar	wml:setvar
wml:go	wml:timer
wml:noop	

The WML2 document type also adds to to the XHTML elements specified the following attributes:

Element	Attributes Added
body	wml:onenterbackward, wml:onenterforward, wml:ontimer, wml:newcontext
html	wml:onenterforward, wml:onenterbackward, wml:ontimer, wml:use-xml-fragments
img	wml:localsrc
input	wml:emptyok, wml:format, wml:name
meta	wml:forua
option	wml:onpick
p	wml:mode
select	wml:iname, wml:ivalue, wml:name, wml:value
textarea	wml:emptyok, wml:format, wml:name

The WML2 document type includes the u element for backward compatibility.

Conformance Guidelines

The specification adopts existing W3C specifications [XHTMLBasic] and [XHTMLMod], but does not itself specify conformance requirements for [XHTML].

Conformance to this specification is defined in the Static Conformance Requirements (SCR) table in Appendix D.

* MERGEFORMAT * MERGEFORMAT * MERGEFORMAT * MERGEFORMAT * MERGEFORMAT * MERGEFORMAT * MERGEFORMAT * MERGEFORMAT * MERGEFORMAT

5. User Agent Behaviour (Normative)

5.1. User Agent Behaviour and Features in WAE

The specification covers the elements of the language (see section 6), the use of style with WML (see section 7) and other aspects of user agent behaviour necessary to implement WML as part of the Wireless Application Environment.

However WML2 depends on user agent behaviour and features described outside of this specification. To fully understand, implement and use WML2, this specification must be considered in conjunction with the Wireless Application Environment specification [WAE] and other specifications referenced in both this specification and [WAE].

5.2. User Agent Context

The *user agent context* consists of variables and navigation history.

5.2.1. Variables

A *WML variable* is a name-value pair, where the value is any string in the document character set.

This section describes WML variables in the user agent context. The user agent MUST implement WML variables.

WML variables can be used in the place of strings and are substituted at run-time with their current value.

A variable is said to be *set* if it has a value not equal to the empty string. A variable is *not set* if it has a value equal to the empty string, or is undefined in the current user agent context.

5.2.2. Navigation History

The *navigation history* is a logical stack of the resources in the navigational path the user traversed to arrive at the current resource.

This section describes the navigation history maintained by the user agent. The user agent MUST implement the navigation history model and support all the operations on it defined below.

WML2 includes a simple navigational history model that allows the author to manage backward navigation in a convenient and efficient manner. The user agent history is modelled as a stack of entries that represent the resources in the navigational path the user traversed to arrive at the current location. The stack is configured temporally, such that the newest entry is at the top of the stack and the oldest entry is at the bottom of the stack. Three operations may be performed on the history stack:

- Reset - the history stack may be reset to a state where it only contains the current entry. See the `wml:newcontext` attribute(section 6.2.1) and `newcontext` attribute (section 6.2.3) for more information.
- Push - a new entry is pushed onto the history stack as an effect of forward navigation.
- Pop - the current entry (top of the stack) is popped as a result of backward navigation.

As each body/card is accessed via an explicitly specifiedURI, (e.g., via the `href` attribute in `wml:go` element,) an entry for the body/card is added to the history stack even if it is identical to the most recent entry. At a minimum, each entry must record the resource request information that comprises the absolute URI of the body/card, the method (get or post) used to access the body/card, the value of any postfields, any XHTML form data name-value pairs, and any request headers. Document content is not stored in the history. Variable references are never stored in the history. Any variable references in the resource request information must be replaced with the current value of the variables before the entry is added to the stack.

The user agent MUST return the user to the previous body/card specified in the history if a prev task is executed (see section 5.3.2). The execution of prev task pops the current entry from the history stack.

If the body/card is part of a document that was originally fetched using an HTTP post method, and the user agent MUST re-fetch the document to establish the body/card, then the user agent MUST reissue any post data associated

with fetching the original document. The post data values must be the same values used to fetch the original document. Refer to section 5.3.2 for more information on the semantics of the prev task. See [CACHE] for more information on caching semantics.

5.3. Navigation Reference Processing Model

A *task* is a navigational directive executed by the user agent in response to an event such as the activation of a `wml:do` element or the expiration of a timer. The *navigation reference processing model* defines the navigational tasks and the operations those tasks perform on the user agent context.

The user agent MUST implement all the tasks defined in this section. In addition, the user agent MUST implement the task failure behaviour.

The following tasks are defined in this section:

- The go task
- The prev task
- The noop task
- The refresh task

5.3.1. The Go Task

The user agent MUST perform the following steps to execute a go task:

1. If the originating task contains `wml:setvar` elements, the variable name and value in each `wml:setvar` element is converted into a simple string by substituting all referenced variables. The resulting collection of variable names and values is stored in temporary memory for later processing. See section 5.5.1.1 for more information on variable substitution.
2. The target URI is identified and fetched by the user agent. The `href` attribute value is first converted into a simple string by substituting all referenced variables.
3. The access control parameters for the fetched document are processed as specified in section 6.16.
4. The destination is located using the fragment identifier specified in the URI.
   a) If a fragment identifier is not specified as part of the URI, the first body/card in the document is the destination.
   b) If the `wml:use-xml-fragments` attribute on the `html` element is `true` (default state)
      i. If a fragment identifier is specified as part of the URI and the document contains a matching element (`id` attribute of the element is identical to the fragment identifier), the enclosing body/card is the destination. If the matching element is a `body` or `wml:card` that body/card is the destination. If the destination is within the current body/card, the user agent should ensure the matching element is visible to the user.
      ii. If a matching element cannot be found, the first segment in the document is the destination.
   c) If the `wml:use-xml-fragments` attribute on the `html` element is `false`
      i. If a fragment identifier is specified as part of the URI and the document contains a matching body/card (id attribute of the body/card is identical to the fragment identifier), that body/card is the destination.
      ii. If a matching body/card cannot be found, the first body/card in the document is the destination.
5. The variable assignments resulting from the processing done in step #1 (the `wml:setvar` element) are applied to the current user agent context.

6. If the destination contains a `newcontext` or `wml:newcontext` attribute, a new user agent context is created as described in sections 6.2.1 and 6.2.3.

7. Create a stack entry. If the destination contains any forms, initialise each form according to section 5.4.2 Form Initialisation, then store all locally-scoped form control variables in the entry.

8. An entry referring to the destination is pushed onto the history stack.

9. If the destination specifies an enterforward intrinsic event binding, the task associated with the event binding is executed and processing stops. See section 5.6.1 for more information.

10. If the destination contains a `wml:timer` element, the timer is started as specified in section 6.16.11.

11. The destination is displayed using the current variable state and processing stops.

5.3.2. The Prev Task

The user agent MUST perform the following steps to execute a prev task:

1. If the originating task contains `wml:setvar` elements, the variable name and value in each `wml:setvar` element is converted into a simple string by substituting all referenced variables. The resulting collection of variable names and values is stored in temporary memory for later processing. See section 5.5.1.1 for more information on variable substitution.

2. The history stack is popped. The target URI is the top of the history stack. The target URI is identified and fetched by the user agent. The processing that occurs if there is no previous entry in the history stack is implementation dependent.

3. The destination is located using the fragment identifier specified in the URI.

    a) If a fragment identifier is not specified as part of the URI, the first body/card in the document is the destination.

    b) If the `wml:use-xml-fragments` attribute on the `html` element is `true`

        i. If a fragment identifier is specified as part of the URI and the document contains a matching element (`id` attribute of the element is identical to the fragment identifier), the enclosing body/card is the destination. If the matching element is a `body` or `wml:card` , that body/card is the destination. If the destination is within the current body/card, the user agent should ensure the matching element is visible to the user.

        ii. If a matching element cannot be found, the first body/card in the document is the destination.

    c) If the `wml:use-xml-fragments` attribute on the `html` element is `false`

        i. If a fragment identifier is specified as part of the URI and the document contains a matching body/card (`id` attribute of the body/card is identical to the fragment identifier), that body/card is the destination.

        ii. If a matching body/card cannot be found, the first body/card in the document is the destination.

4. The variable assignments resulting from the processing done in step #1 (the `wml:setvar` element) are applied to the current user agent context.

5. If the destination contains any locally scoped form data, it is refreshed with the form data stored in the entry at the top of the history stack.

6. If the destination specifies an enterbackward intrinsic event binding, the task associated with the event binding is executed and processing stops. See section 5.6.1 for more information.

7. If the destination contains a `wml:timer` element, the timer is started as specified in section 6.16.

8. The destination is displayed using the current variable state and processing stops.

5.3.3. The Noop Task

No processing is done for a noop task.

5.3.4. The Refresh Task

The user agent MUST perform the following steps to execute a refresh task:

1. For each `wml:setvar` element, the variable name and value in each `wml:setvar` element is converted into a simple string by substituting all referenced variables. See section 5.5.1.1 for more information on variable substitution.
2. The variable assignments resulting from the processing done in step #1 (the `wml:setvar` element) are applied to the current user agent context.
3. If the destination contains any locally scoped form data, it is refreshed with the form data stored in the entry at the top of the history stack.
4. If the current body/card contains a `wml:timer` element, the timer is started as specified in section 6.16.11.
5. The current body/card is re-displayed using the current variable state and processing stops.

User-visible effects of the state changes (e.g., a change in the screen display resulting from the changes of referenced variables) SHALL occur during the processing of the refresh task. A refresh and its user-visible effects SHALL occur even if the elements have no `wml:setvar` elements, given that context may change by other means (e.g., timer).

5.3.5. Task Execution Failure

If a task fails to fetch its target resource or the access control restrictions prevent a successful inter-body/card transition, the user agent MUST notify the user and take the following actions:

- The invoking body/card remains the current body/card.
- No changes are made to the browser context, including any pending variable assignments or newcontext processing.
- No intrinsic event bindings are executed.

5.4. Form Processing Reference Model

A *form* is a part of a WML2 document consisting of a container element enclosing structured content and one or more of the elements `input`, `select`, `textarea`. These elements are called form control elements. Form control elements may be placed within a `form` element, or they may be placed directly within a `wml:card` element. Throughout this specification, unless stated otherwise, *form* refers to either a `form` element containing form control elements, or a `wml:card` element containing form control elements.

This section describes the processing of forms.

The user agent MUST implement the processing of forms.

5.4.1. Overview of Form Processing Model

The form control elements are those from the Basic Forms Module of XHTML. To allow authors to group logically related content and controls, the `optgroup` and `fieldset` elements from the XHTML Forms Module (a superset of Basic Forms) are also included.

The WML2 Form Processing Model describes the life cycle of a form. The stages are:

1. Initialisation

(See 5.4.2 for how the user agent must process a form before displaying a body/card containing that form.)

2. Interaction

   (See 5.4.3 for how the user agent must handle user interactions with the form.)

3. Committing Data

   (See 5.4.4 for how the user agent must commit the form data to the user agent context before leaving a form.)

4. Submission

   (See 5.4.5 for how the user agent must submit a form.)

In WML2, each form control is associated with a *form control variable.* A form control variable is a WML variable. The control's value is stored in this variable for later processing.

For compatibility with XHTML forms, form control variables have scope. The declaration of the form control variable determines its scope. There are two types of scope: global and local (form scope). Locally scoped variables are only visible within the form in which they are declared. Globally scoped variables are visible throughout the user agent context. See section 5.5.1.4 for details on variable scoping.

5.4.2. Form Initialisation

The form initialisation phase of the WML2 Form Processing Model determines the initial values of the form's controls, that is, the values they have upon initial display of the document containing the form. After the controls' initial values are determined, they are committed to form control variables. These initial values are also saved for use during form reset.

Prior to displaying a `wml:card` or a `body` element containing a form, the user agent MUST initialise each form in the following manner:

- The user agent MUST initialise each control within the form according to section 5.4.2.1, Form Control Initialisation.
- The user agent MUST process the controls in the order in which they appear in the form container element.
- The user agent MUST store the initial value of every control, for use during form reset.

The user agent MUST perform form initialisation when a body or card is reached through either forward or backward navigation or refresh. The user agent MUST perform form initialisation at the appropriate step in the reference processing model. See section 5.3 for details.

5.4.2.1. Form Control Initialisation

The user agent initialises each form control within the form as it prepares the form for display. Initialisation of a form control means determining its initial value, and storing that value in the form control variable.

Prior to form display, the user agent MUST initialise each form control in the following manner:

- The user agent MUST process text input controls according to section 5.4.2.2. Text controls are `input` elements with `type="text"` and `type="password"` and `textarea` elements.
- The user agent MUST process menu controls according to section 5.4.2.3. Menu controls are `select` elements together with their `option` children.
- The user agent MUST process checkbox controls according to section 5.4.2.4. Checkbox controls are `input` elements with `type="checkbox"`.
- The user agent MUST process radio button controls according to section 5.4.2.5. Radio button controls are `input` elements with `type="radio"`.

- The user agent MUST process submit button controls according to section 5.4.2.6. Submit button controls are `input` elements with `type="submit"`.
- The user agent MUST process reset button controls according to section 5.4.2.7. Reset button controls are `input` elements with `type="reset"`.
- The user agent MUST process hidden controls according to section 5.4.2.8. Hidden controls are `input` elements with `type="hidden"`.

5.4.2.2. Text Input Control Initialisation

The user agent MUST initialise a text input control in the following manner:

- If the `wml:name` attribute is assigned, and the control is an `input` element, set the initial value of the control as follows:

  1. The `wml:name` attribute names the form control variable. If the value of the form control variable conforms to the input mask, as defined by the `wml:format` and `wml:emptyok` attributes or their CSS equivalents, the initial value of the control is the value of the control variable. See section 6.5.2 for definition of input mask and details of `wml:format` and `wml:emptyok` attributes.
  2. If the value of the form control variable does not conform to the input mask, it is ignored and the initial value of the control is determined by the `value` attribute:
     a) If the value of the `value` attribute conforms to the input mask, the initial value of the control is that attribute value.
     b) If the value of the `value` attribute does not conform to the input mask, the `value` attribute is ignored and the initial value is the empty string.
     c) If the `value` attribute is not specified, the initial value of the control is the empty string.
  3. Set the control variable equal to the initial value of the control.
  4. Pre-load the initial value into the form control.

- If the `wml:name` attribute is assigned, and the control is a `textarea` element, set the initial value of the control as follows:

  1. The `wml:name` attribute names the form control variable. If the value of the form control variable conforms to the input mask, as defined by the `wml:format` and `wml:emptyok` attributes or their CSS equivalents, the initial value of the control is the value of the control variable.
  2. If the value of the form control variable does not conform to the input mask, it is ignored and the initial value of the control is determined by the `textarea` element:
     a) If the content of the `textarea` element conforms to the input mask, the initial value of the control is the content of the element.
     b) If the content of the `textarea` element does not conform to the input mask, the content of the element is ignored and the initial value is the empty string.
     c) If the `textarea` element is an empty element, the initial value of the control is the empty string.
  3. Set the control variable equal to the initial value of the control.
  4. Pre-load the initial value into the form control.

- If the `name` attribute is assigned, and the control is an `input` element, set the initial value of the control as follows:

1. The initial value of the control is determined by the `value` attribute and the input mask, as defined by the `wml:format` and `wml:emptyok` attributes or their CSS equivalents:

    a) If no input mask is specified, the initial value of the control is the value of the `value` attribute. If the `value` attribute is not specified, the initial value of the control is the empty string

    b) If the input mask is specified and the value of the `value` attribute conforms to the input mask, the initial value of the form control is that value.

    c) If the input mask is specified and the value of the `value` attribute does not conform to the input mask, the initial value of the control is the empty string.

2. Set the control variable equal to the initial value. (The name of the control variable is the value of the `name` attribute.)

3. Pre-load the initial value into the form control.

- If the `name` attribute is assigned, and the control is a `textarea` element, set the initial value of the control as follows:

    1. The initial value of the control is determined by the content of the `textarea` element and the input mask, as defined by the `wml:format` and `wml:emptyok` attributes or their CSS equivalents:

        a) If no input mask is specified, the initial value of the control is the content of the element. If the element is an empty element, the initial value of the control is the empty string.

        b) If the input mask is specified and the content of the element conforms to the input mask, the initial value of the control is the content of the element.

        c) If the input mask is specified and the content of the element does not conform to the input mask, the initial value of the control is the empty string.

    2. Set the control variable equal to the initial value. (The name of the control variable is the value of the `name` attribute.)

    3. Pre-load the initial value into the form control.

- If both the `wml:name` attribute and the `name` attribute are assigned, `wml:name` takes precedence, and the `name` attribute MUST be ignored.

- If neither the `wml:name` attribute nor the `name` attribute is assigned:

    1. The control does not have a corresponding form control variable. This control cannot be *successful* as defined by [HTML4].

    2. The initial value of the control is the value of the `value` attribute, except when the form control element is `textarea`. For `textarea`, the initial value is the content of the element. If the `value` attribute is not assigned, or the form control has no content, then the initial value is the empty string.

    3. Pre-load the initial value into the form control.

5.4.2.3. Menu Control Initialisation

The pre-selection of option elements includes an operation named *validate*. This operates on a value, and determines if that value is a legal option index (or indices for a multiple-selection menu). The operation consists of the following steps:

1. Remove all non-integer indices from the value.

2. Remove all out-of-range indices from the value, where out-of-range is defined as any index with a value greater than the number of options in the menu or with a value less than one.

3. Remove duplicate indices.

Note that an invalid index will result in an *empty* value.

The user agent MUST initialise a menu control in the following manner:

- If any of the `wml:name`, `wml:iname`, `wml:value` or `wml:ivalue` attributes is assigned, the `selected` attribute of the `option` element is ignored. Set the initial value of the control as follows:
  - If the control only permits a single selection:
    1. Determine the default option index:
       a) If the `wml:iname` attribute is specified, it names the variable to use to determine the default option index. The default index is the value of named variable.
       b) If the default option index has not yet been determined and the `wml:ivalue` attribute is specified, the default option index is the value of the `wml:ivalue` attribute.
       c) If the default option index has not yet been determined and the `wml:name` attribute is specified, it names the variable to use to determine the default option index. The default index is the index of the option whose `value` attribute equals the value of the named variable.
       d) If the default option index has not yet been determined and the `wml:value` attribute is specified, the default option index is the index of the option whose `value` attribute equals the value of the `wml:value` attribute.
       e) If no option has been pre-selected, the default option index is one.
    2. Set the control variable(s):
       a) If the `wml:iname` attribute is specified, set the named variable with the default option index.
       b) If the `wml:name` attribute is specified, set the named variable with the value of the `value` attribute on the `option` element at the default option index. If the `value` attribute is not specified, set the variable to the empty string.
    3. Pre-select the option at the default option index.
  - If the control permits multiple selections:
    1. Determine the default option indices:
       a) If the `wml:iname` attribute is specified, it names the variable used to determine the default option indices. The value of the named variable is a semicolon-delimited list of the indices of the pre-selected `option` elements.
       b) If the default option indices have not yet been determined and the `wml:ivalue` attribute is specified, its value is used to determine the default option indices. The attribute value is a semicolon-delimited list of the indices of the pre-selected `option` elements.
       c) If the default option indices have not yet been determined and the `wml:name` attribute is specified, it names the variable used to determine the default option indices. The value of the named variable is a semicolon-delimited list of the values of the pre-selected `option` elements' `value` attributes. The default indices are the corresponding indices.
       d) If the default option indices have not yet been determined and the `wml:value` attribute is specified, its value is used to determine the default option indices. The attribute value is a semicolon-delimited list of the values of the pre-selected `option` elements' `value` attributes. The default indices are the corresponding indices.

2. Set the control variable(s):

   a) If the `wml:iname` attribute is specified, set the named variable with the default option indices. If there are no default options, set the variable to the empty string.

   b) If the `wml:name` attribute is specified, set the named variable to a semicolon-delimited list of the following values: for each index in the default indices greater than zero, the value of the `value` attribute on the `option` element at the index. If there are no default options, or if all selected `option` elements contain an empty `value` attribute, set the variable to the empty string.

3. Pre-select options specified by the default option indices:

   a) Deselect all options.

   b) For each index greater than zero, select the option specified by the index.

- If the `name` attribute is assigned, it names the control variable. Set the initial value of the control as follows:
  - If the control only permits a single selection:
    1. If exactly one `option` has the `selected` attribute assigned, that option is the pre-selected option. If more than one `option` has the `selected` attribute assigned, the first such option is the pre-selected option.
    2. Set the control variable equal to the value of the pre-selected `option` element's `value` attribute. If the `value` attribute is not specified, set the variable to the contents of the `option` element. If the element is empty, set the variable to the empty string.
    3. Select the menu item corresponding to the pre-selected option.
  - If the control permits multiple selections:
    1. The pre-selected options are those option elements with their `selected` attribute assigned. If no `option` has the `selected` attribute assigned, there are no pre-selected options.
    2. Set the control variable equal to a string consisting of the semicolon-delimited list of the values of the pre-selected `option` elements' `value` attributes. If the `value` attribute of any pre-selected `option` is not specified, substitute the contents of the element, but ignore the option if the element is empty. If there are no pre-selected options, set the control variable to the empty string.
    3. Select the menu items corresponding to the pre-selected options.
- Assigning any of `wml:name`, `wml:iname`, `wml:value` or `wml:ivalue` takes precedence over `name`; the `name` attribute and `option` selections expressed via the `selected` attribute MUST be ignored whenever any of `wml:name`, `wml:iname`, `wml:value` or `wml:ivalue` is assigned.

5.4.2.4. Checkbox Control Initialisation

The user agent MUST initialise a checkbox control in the following manner:

- Initialise the control as specified for a multiple-selection select, where each checkbox control that shares the same control name is mapped to a subordinate `option` element. As part of that mapping, the `checked` attribute is mapped to the `option` element's `selected` attribute. See section 5.4.2.3.

5.4.2.5. Radio Button Control Initialisation

The user agent MUST initialise a radio button control in the following manner:

- Initialise the control as specified for a single-selection select, where each radio button control that shares the same control name is mapped to a subordinate `option` element. As part of that mapping, the `checked` attribute is mapped to the `option` element's `selected` attribute. See section 5.4.2.3.

5.4.2.6. Submit Button Control Initialisation

The user agent MUST initialise a submit button control in the following manner:

- The name of the control variable is the value of the `name` attribute. Set the control variable equal to the value of the `value` attribute.

5.4.2.7. Reset Button Control Initialisation

The user agent MUST initialise a reset button control in the following manner:

- No initialisation is performed.

5.4.2.8. Hidden Control Initialisation

The user agent MUST initialise a hidden control in the following manner:

- If the `wml:name` attribute is assigned:

  1. The `wml:name` attribute names the control variable. The initial value of the control is the value of the control variable. If the control variable is not set, the initial value of the control is the value of the `value` attribute. If the `value` attribute is not specified, the initial value is the empty string.
  2. Set the control variable equal to the initial value of the control.

- If the `name` attribute is assigned:

  1. The name of the control variable is the value of the `name` attribute. The initial value of the control is the value of the `value` attribute. If the `value` attribute is not specified, the initial value is the empty string.
  2. Set the control variable equal to the initial value of the control.

- If both are assigned:

  1. The `wml:name` attribute takes precedence, and the `name` attribute MUST be ignored.

- If neither attribute is assigned:

  1. The control does not have a corresponding form control variable. The control cannot be *successful* as defined by [HTML4].

     a) The name of the control variable is the value of the `name` attribute. Set the control variable to the value of the `value` attribute.

5.4.3. Form Interaction

The interaction phase of the WML2 Form Processing Model determines user agent behaviour as the user interacts with form controls within a form. The interaction phase updates the control's current value based on user interactions. If user interaction with a form control modifies its current value, the user agent performs an update to the value presented to the user as well as to the form control variable.

Once a form has been displayed, the user agent MUST handle user interaction with the individual controls within the form according to section 5.4.3.1, Form Control Interaction. The user agent MUST continue processing user interaction until the execution of the next task.

The user agent MAY re-display the current body or card when form control variables are updated, but authors must not rely on this behaviour.

5.4.3.1. Form Control Interaction

If user interaction with a form control modifies the control's current value, the user agent MUST handle that interaction as follows:

- The user agent MUST process text input controls according to section 5.4.3.2. Text controls are input elements with `type="text"` and `type="password"` and the `textarea` element.
- The user agent MUST process menu controls according to section 5.4.3.3. Menu controls are `select` elements together with their `option` children.
- The user agent MUST process checkbox controls according to section 5.4.3.4. Checkbox controls are `input` elements with `type="checkbox"`.
- The user agent MUST process radio button controls according to section 5.4.3.5. Radio button controls are `input` elements with `type="radio"`.
- The user agent MUST process submit button controls according to section 5.4.3.6. Submit button controls are `input` elements with `type="submit"`.
- The user agent MUST process reset button controls according to section 5.4.3.7. Reset button controls are `input` elements with `type="reset"`.
- The user agent MUST process hidden controls according to section 5.4.3.8. Hidden controls are `input` elements with `type="hidden"`.

5.4.3.2. Text Input Control Interaction

The user agent MUST handle interaction with a text input control in the following manner:

- When the user attempts to commit input, the user agent MUST validate the input against the input mask, as defined by the `wml:format` and `wml:emptyok` attributes or their CSS equivalents:

    1. If no input mask is specified, the user agent MUST set the control variable to the control's current value.
    2. If the control's current value conforms to the input mask, the user agent MUST set the control variable to that va lue.
    3. If the control's current value does not conform to the input mask, the user agent MUST NOT commit that input and MUST notify the user that the input was rejected and allow the user to resubmit new input. In this case, the control variable MUST NOT be modified.

- The user agent MAY validate user input at any other appropriate time, e.g., when control focus is lost or with each character entered.

5.4.3.3. Menu Control Interaction

Interaction with a menu control is the selection or de-selection of an `option` element. The user agent MUST handle this interaction by updating the control variable(s) in the following manner:

- If the `wml:name` or `wml:iname` attributes were used to name the control variable(s):

    1. If the control only permits a single selection:

        a) If the `wml:name` attribute is specified, set the named variable to the value of the `value` attribute on the selected `option` element. If the `value` attribute is not specified, set the variable to the empty string.

        b) If the `wml:iname` attribute is specified, set the named variable to the index of the selected option. Index numbering begins at one and increases monotonically.

2. If the control permits multiple selections:

    a) If the `wml:name` attribute is specified, set the named variable to a string consisting of the semicolon-delimited list of the values of the selected `option` elements' `value` attributes. If any `value` attribute is not specified or is empty, ignore that option. If there are no selected options, or if all selected `option` elements contain an empty `value` attribute, set the control variable to the empty string.

    b) If the `wml:iname` attribute is specified, set the named variable to a string consisting of the semicolon-delimited list of the indices of the selected `option` elements. If there are no selected options, set the control variable to the empty string.

- If the `name` attribute was used to name the control variable:

1. If the control only permits a single selection:

    a) Set the control variable equal to the value of the selected `option` element's `value` attribute. If the `value` attribute is not specified, set the variable to the contents of the selected `option` element, or to the empty string if the element is empty.

2. If the control permits multiple selections:

    a) Set the control variable equal to a string consisting of the semicolon-delimited list of the values of the selected `option` elements' `value` attributes. If the `value` attribute of any selected `option` is not specified, substitute the contents of the element, but ignore the `option` if the element is empty. If there are no selected options, set the control variable to the empty string.

Controls that permit multiple selections may result in a control variable with a value that is a semicolon-delimited list (e.g., "dog;cat"). This is not an ordered list and the user agent MAY construct the list in any order that is convenient. Authors must not rely on a particular value ordering. The user agent MUST ensure that the `wml:iname` variable value contains no duplicate index values. The `wml:name` variable value MUST contain duplicate values in the situation where multiple selected option elements have the same value. The `wml:name` result MUST NOT contain empty values (e.g., "cat;;dog" is illegal).

However, in all cases, the user agent MUST NOT display user-visible effects as a result of updating `wml:name` and `wml:iname` variables, except when there is an explicit refresh task (see section 5.3). The user agent MUST update `wml:name` and `wml:iname` variables (if specified) for each select element in the card before each and every task invocation.

5.4.3.4. Checkbox Control Interaction

The user agent MUST handle interaction with a checkbox control in the following manner:

- Handle the control as specified for a multiple-selection select, where each checkbox control that shares the same control name is mapped to a subordinate `option` element. As part of that mapping, the checked attribute is mapped to the `option` element's selected attribute. See section 5.4.3.3.

5.4.3.5. Radio Button Control Interaction

The user agent MUST handle interaction with a radio button control in the following manner:

- Handle the control as specified for a single-selection select, where each radio button control that shares the same control name is mapped to a subordinate option element. As part of that mapping, the checked attribute is mapped to the option element's selected attribute. See section 5.4.3.3.

5.4.3.6. Submit Button Control Interaction

The user agent MUST handle interaction with a submit button control in the following manner:

- Submit the form as specified in section 5.4.5, Form Submission.

5.4.3.7. Reset Button Control Interaction

The user agent MUST handle interaction with a reset button control in the following manner:

1. Set the current value of each control within the current form to its initial value, as in [HTML4]
2. Update each form control variable to reflect the new current value of the control.

5.4.3.8. Hidden Control Interaction

By definition, no interaction with hidden controls is possible.

5.4.4. Committing Form Data

Before navigation away from a body or card containing a form, the user agent MUST commit the current value of each form control to its form control variable, as specified in section 5.4.3, Form Interaction. This applies to both form submission and the execution of any task.

5.4.5. Form Submission

The form submission phase of the WML2 Form Processing Model determines how the form data set is submitted to the server for processing. This section describes submission of forms whose container element is the `form` element. Submission of forms whose container element is the `wml:card` element is accomplished with a go task. See section 6.16.4 for more information. The functionality of the go task is a superset of XHTML form submission. To submit a form, the user agent must logically map form submission into the execution of a go task. It is not necessary for the user agent to physically construct a go task element.

The user agent MUST submit a form for processing according to the following steps:

1. Construct the form data set by executing the steps of [HTML4] section 17.13.3, Processing form data. The user agent MUST process all successful controls, including those bound to global variables.
2. Construct a go task. Map the `form` element's `action` attribute to the go task's `href` attribute. Map the `form` element's `method` and `enctype` attributes to the go task's `method` and `enctype` attributes, respectively.
3. For each name-value pair in the form data set, insert a `postfield` element as a child of the go task. Map the control name to the `postfield` element's `name` attribute and the control value to its `value` attribute.
4. Execute the go task.

5.5. Attribute Expression Syntax

The attribute expression syntax is used in parameterisation of the values of certain types of attributes, which are interpreted as expressions and evaluated at run-time by the user agent.

With the attribute expression syntax, all WML content can be parameterised, allowing the author a great deal of flexibility in creating documents with improved caching behaviour and better perceived interactivity. The values of certain types of attributes are interpreted as expressions and evaluated at run-time. The attribute expression syntax in conjunction with the `wml:getvar` element (see section 6.16.10) provides authors a simple yet powerful mechanism to parameterise content.

Only the values of attributes of type `CDATA` are interpreted as expressions. This includes attributes with the XHTML derived attribute types Character, Charset, Charsets, Color, ContentType, ContentTypes, Datetime, Length, MediaDesc, MultiLength, Number, Pixels, Script, Text, URI and URIs [XHTMLMod]. The result of evaluating an expression is always a string, in the character set of the document. The document that results from evaluating all expressions within a source document's `CDATA` attributes and processing all `wml:getvar` elements is called the *presentation document*.

For a document whose type is XHTML Mobile Profile, the user agent MUST process the document assuming no attribute expression syntax.

For a document whose type is WML2, the user agent MUST process the document according to the attribute expression syntax.

For identification of the document type, see section 5.7.

If the type of a document cannot be determined, the user agent MAY process the document assuming no attribute expression syntax. In this case, the user agent MAY display an appropriate warning message to the user.

5.5.1. Attribute Expression Syntax Processing

The values of variables can be substituted into certain attribute values in WML elements, using a variable reference. When a variable reference is resolved, it results in the substitution of the variable's value. Substitution does not affect the current value of the variable and is defined as a string substitution operation. If an undefined variable is referenced, it results in the substitution of the empty string. The `wml:getvar` element can be used to substitute the values of variables into the text (`#PCDATA`) of a document (see section 6.16.10 for more information).

The source document is processed according to the variable expression syntax as follows:

For each CDATA attribute type defined in section 5.5, evaluate the value of the attribute as an expression in the attribute expression syntax. Each expression is evaluated from left to right. Evaluation of an expression consists of the following steps:

- Initially set the result string to the empty string.
- For each literal character, append it to the result string.
- For each valid variable reference, append the value of the variable to the result string. Variable references are described in section 5.5.1.1.
- For each literal dollar sign escape sequence, append the '$' character to the result string. The dollar sign escape sequence is described in section 5.5.1.3.

These operations are described in the following sections.

5.5.1.1. Variable References

WML variable names consist of a US-ASCII letter or underscore followed by zero or more letters, digits or underscores. Any other characters are illegal. Variable names are case sensitive.

The following description of the variable reference syntax uses the Extended Backus-Naur Form (EBNF) notation established in [XML].

```
varref    ::= ( "$" varname ) | ( "$(" varname ( conv )? ")" )
varname   ::= ( "_" | alpha ) ( "_" | alpha | digit )*
conv      ::= ":" ( "escape" | "noesc" | "unesc" )
alpha     ::= [a-zA-Z]
digit     ::= [0-9]
```

Variable references may use parentheses for clarity. Parentheses are required anywhere the end of a variable name cannot be inferred from the surrounding context.

For example:

```
This is a $(var).
```

In this example, a reference to the variable "direction" requires parentheses:

```
You should navigate $(direction)ward.
```

In the next example, the variable name “var” can be inferred, so parentheses are not needed:

```
Here’s another reference to $var.
```

Other legal variable forms:

```
$_X $X32 $Test_9A
```

The value of variables can be converted into a different form as they are substituted. A conversion can be specified in the variable reference following the colon. The following table summarises the current conversions.

Table 5-1. Conversions

Conversion	Effect
noesc	No change to the value of the variable.
escape	URI-escape the value of the variable.
unesc	URI-unescape the value of the variable.

The use of a conversion during variable substitution does not affect the actual value of the variable. URI-escaping is detailed in [RFC2396]. All lexically sensitive characters defined in WML must be escaped, including all characters not in the `unreserved` set specified by [RFC2396].

If no conversion is specified, the variable is substituted using the conversion format appropriate for the context. All attributes defined as type URI or URIs [XHTMLMod], default to `escape` conversion. For attributes with other types, no conversion is done. Specifying the `noesc` conversion disables context sensitive escaping of a variable.

For example:

```
This is an escaped $(var:escape).
This is an unescaped $(var:unesc).
```

5.5.1.2. Parsing the Variable Reference Syntax

The variable reference syntax (e.g., `$X`) is parsed after all XML parsing is complete. In XML terminology, variable substitution is done after the *XML processor* has parsed the document and provided the resulting parsed form to the *XML application*. In the context of this specification, the WML parser and user agent is the XML application.

This implies that all variable reference syntax is parsed *after* the XML constructs, such as tags and entities, have been parsed. In the context of variable parsing, all XML syntax has a higher precedence than the variable reference syntax, e.g., entity substitution occurs before the variable reference syntax is parsed. The following examples are identical references to the variable named X:

```
$X
&#x24;X
$&#x58;
&#36;&#x58;
```

5.5.1.3. The Dollar-sign Character

A side effect of the parsing rules is that the literal dollar sign must be encoded with a pair of dollar sign entities in any attribute values. A single dollar-sign entity, even specified as `&#x24;`, results in a variable substitution.

In order to include a '$' character in a WML document, it must be explicitly escaped. This can be accomplished with the following syntax:

```
$$
```

Two dollar signs in a row are replaced with a single '$' character. For example:

```
This is a $$ character.
```

This would be evaluated as:

```
This is a $ character.
```

To include the '$' character in URI-escaped strings, specify it with the URI-escaped form:

```
%24
```

5.5.1.4. Variable Scoping

One way that variables are used in WML is as *form control variables.* A form control variable is a variable that stores the current value of a form control. For compatibility with XHTML forms, form control variables have scope. The declaration of the form control variable determines its scope.

There are two types of scope: global and local (form scope). Locally-scoped variables are only visible within the form in which they are declared. Globally-scoped variables are visible throughout the current document and in all documents processed in the current context.

5.5.1.4.1. Scoped Variable Declarations

Locally-scoped (form-scoped) variables are declared by use of the `name` attribute on a form control element. Globally-scoped variables are declared by use of the `wml:name` attribute on a form control element. A form can contain any mix of locally- and globally-scoped form control variables.

For example, the following form declares a locally-scoped variable `x` and a globally-scoped variable `y`:

```
<form action="...">
 <p>
 <input name="x" value="foo"/>    <!-- locally-scoped var x -->
 <input wml:name="y"/>            <!-- globally-scoped var y -->
 </p>
</form>
```

Form control elements may be placed outside of a `form` element. If such a control names a control variable with the `wml:name` attribute, a globally-scoped form control variable is declared. If such a form control names a control variable with the `name` attribute, the control does not map to a variable (for compatibility with XHTML form semantics).

Note: Variables declared (set) with the `wml:setvar` element are always globally-scoped, regardless of the positioning of the `wml:setvar` element relative to a `form` element.

5.5.1.4.2. Scoped Variable References

Due to form control variable scoping, variable references are evaluated according to their position relative to the `form` element. Within a `form` element, any locally-scoped variable will hide a globally-scoped one with the same name. If none is present, a reference binds to the globally-scoped variable. Outside of a `form` element (either within a `body` or `wml:card` element), a reference always binds to a globally-scoped variable.

For example, assuming the global variable `x` is set:

```
<wml:card>
 <wml:getvar name="x"/>   <!-- refers to global variable x -->
 <form action="...">
  <p>
  <input name="x"/>
```

```
    <wml:getvar name="x"/>  <!-- refers to local variable x -->
    <wml:getvar name="y"/>  <!-- reference not resolved -->
    </p>
   </form>
  </wml:card>
```

If a variable reference names a form control variable, the user agent MUST evaluate the reference as follows:

- If the reference is located within a `form` element, the reference evaluates to the value of the locally-scoped variable with the specified name. If there is no such local variable set, the reference evaluates to the value of the global variable with the specified name. If there is no such global variable set, the reference evaluates to the empty string.
- If the reference is located outside of a `form` element, the reference evaluates to the value of the global variable with the specified name. If there is no such global variable set, the reference evaluates to the empty string.

5.5.1.5. Validation

Within the attribute expression syntax, any string following a single dollar sign ('$') must be treated as a variable reference and validated, unless it is part of an escaped literal dollar sign sequence according to section 5.5.1.3. Each reference must use proper variable name syntax, according to section 5.5.1.1. Specifying any conversion other than those specified in section 5.5.1.1 results in an invalid variable reference.

Values of attributes not evaluated as part of the attribute expression syntax must use an escaped literal dollar sign to prevent the creation of an otherwise valid variable reference.

The document is in error if any variable reference uses invalid syntax or is placed in an invalid location.

An example of an invalid variable reference:

```
<!-- bad variable syntax -->
<img src="$?"/>
```

An example of escaped dollar sign in an attribute of type `CDATA`:

```
<!-- Dollar sign escaped in title attribute -->
<a href="/next" title="Win $$1,000,000!">
```

5.5.1.6. Event Parameters

For WML2, the variable reference syntax add the ability to reference ordered local event parameters, such as those used in [WTA], or event attributes as defined by IDL order in [DOM2EVENT]. An event parameter reference is a variable name consisting of one or more digits, such as `$1` or `$(2:esc)`.

All event parameters are empty except during the presentation evaluation of event listeners, such as tasks in a `wml:do` or `wml:onevent` element. The definition of event parameters is part of the event definition; by default, an event has no parameters.

For variable expressions which comply with the syntactic requirements of section 5.5.1.1 except that the variable name is a positive integer rather than an alphanumeric string, the WML Variable attribute expression language SHALL evaluate variable references as defined in section 5.5.1.1, except that the value of the variable SHALL be defined by the event whose binding the presentation renderer is currently evaluating. If an event defines no parameters, or the current presentation of the document is not evaluating an event listener, the value of the expression SHALL be the empty string.

5.5.1.7. Re-generation of Presentation Document

Before executing a task (see section 5.3) or processing an event binding (see section 5.6), the user agent MUST either re-generate the presentation document, or perform an operation that is indistinguishable from it. This ensures that the evaluation of all expressions reflects the most current user agent context.

5.6. WML2 Event Model

An *event* is a signal to the user agent that some action has taken place. Several WML elements are capable of generating events when the user interacts with them. In addition, the user agent generates events while executing tasks or upon timer expiration.

The following event types are defined in WML:

- *Intrinsic event*: an event generated by the user agent
- *Extrinsic event*: an event sent to the user agent by some external agent

5.6.1. WML Intrinsic Events

Intrinsic events indicate state transitions inside the user agent.

Each event has a target element. The target element is the element within the document to which the event applies. For user interface elements, the target is the element itself. For events generated by the user agent, the target is the current `body` or `wml:card` element. The target element determines where event bindings should be placed.

WML defines the following intrinsic event types.

Table 5-1. WML Intrinsic Events

Event Type	Target Element	Description
`timer`	`body` or `wml:card`	The `timer` event occurs when a timer expires. Timers are specified using the `wml:timer` element (see section 6.16).
`enterforward`	`body` or `wml:card`	The `enterforward` event occurs when the user agent enters a body/card via a go task or any method with identical semantics. This includes body/card entry caused by a script function or user-agent-specific mechanisms, such as a means to directly enter and navigate to aURI.
`enterbackward`	`body` or `wml:card`	The `enterbackward` event occurs when the user agent enters a body/card via a `prev` task or any method with identical semantics. In other words, the `enterbackward` event occurs when the user causes the user agent to navigate into a body/card by using a URI retrieved from the history stack. This includes navigation caused by a script function or user-agent-specific mechanisms.
`pick`	`option`	The `pick` event occurs when the user selects or deselects this item.

5.6.2. WML Extrinsic Events

This specification does not specify any classes of extrinsic events. One example of a WML extrinsic event class may be WTA events [WTA].

5.6.3. Event Bindings

An *event binding* specifies a task to be executed when an event occurs.

The user agent MUST implement bindings of typed and untyped events.

5.6.3.1. Bindings for Typed Events

The specification of an event binding for a typed event takes one of two forms: attribute syntax or element syntax. All the events in Table 5-3 are typed events.

The attribute syntax uses an attribute to specify a URI to be navigated to when the event occurs. This type of event binding is specified in a well-defined element-specific attribute and is the equivalent of a `go` task. For example:

```
<wml:card onenterforward="/url"> <p> Hello </p> </wml:card>
```

This attribute value may only specify a URI.

Table 5-3. Event Bindings for Attribute Syntax

Event Type	Attribute Name
timer	ontimer
enterforward	onenterforward
enterbackward	onenterbackward
pick	onpick

The element syntax is an expanded version of the previous, allowing the author more control over user agent behaviour by giving the ability to specify any task. A `wml:onevent` element is declared within a parent element, specifying the full event binding for a particular event. For example, the following is identical to the previous example:

```
<wml:card>
 <wml:onevent type="enterforward">  <wml:go href="/url"/>
 </wml:onevent>
 <p>
 Hello
 </p>
</wml:card>
```

The user agent MUST treat the attribute syntax as an abbreviated form of the `wml:onevent` element where the attribute name is mapped to the `type` attribute of the `wml:onevent` element.

An event binding is scoped to the element in which it is declared, e.g., an event binding declared in a `body` or `wml:card` is local to that card or body. Any event binding declared in an element is active only within that element. If the event binding element specifies an event type which does not apply to its parent element, the user agent MUST ignore the event binding. Conflicting event bindings within an element are an error.

5.6.3.2. Bindings for Untyped Events

Events targeted at `do` elements are untyped, or anonymous, and cannot be bound using the event binding forms specified in the previous section.

The `wml:do` element therefore uses an abbreviated syntax for specifying event bindings – the task element is placed directly within the `do` element. The `wml:onevent` container element is not used. For example:

```
<wml:do type="accept>
  <wml:go href="url"/>
</wml:do>
```

The only way to bind a task to an event targeted at a `wml:do` element is to place the task element within the `wml:do` element as shown above.

5.6.3.3. Event Binding Scope

Event bindings may be declared within the `body` or `wml:card` elements, and may also be declared at the document level, within the `html` element:

- Card-level: the event binding may appear inside a `body` or `wml:card` element and specify event-processing behaviour for that particular card or body.

- Document-level: the event binding may appear inside the `html` element and specify event-processing behaviour for each body/card in the document. A document-level event binding is equivalent to specifying the event binding in each body/card.

A card-level event binding overrides a document-level event binding if they both specify the same event. A card-level `wml:onevent` element will override a document-level `wml:onevent` element if they both have the same `type`.

For example, in the following document the event handler in the card overrides the one specified in the document root although the two have been defined using different syntax:

```
<html>
 ...
 <wml:onevent type="enterbackward">
   <wml:go href="aaa"/>
 </wml:onevent>
 <wml:card onenterbackward="bbb">
  <p>Hello</p>
 </wml:card>
</html>
```

For a given body/card, the *active* event bindings are defined as the event bindings specified in the body/card that do not bind a `noop` task, plus any event bindings specified in the document root element not overridden in the card or binding a noop task. Overridden event bindings, event bindings defined in other cards, and event bindings that bind a noop task are considered *inactive*.

If a card-level binding overrides a document-level binding and the card-level binding specifies a noop task, the event binding for that event will be completely masked. In this situation, the card- and document-level bindings will be ignored and no side effects will occur on delivery of the event. In other words, both the card-level and the document-level bindings are considered inactive in such a case.

If a card-level binding or document-level binding specifies a `noop` task but does not override and is not overridden by another binding, then the binding for that event will also be masked and similarly ignored with no side effects.

In the following example, a document-level event binding indicates that a go task should execute on receipt of a timer expiration event. The first card inherits the event binding specified in the `html` element. The binding will be active in that card. The second card overrides the document-level event binding with a noop task. The event binding will therefore be inactive when the second card is displayed. The third card overrides the document-level event binding, replacing it with an alternate binding and causing the user agent to execute a different go task if the timer expiration event occurs while the third card is displayed.

```
<html>
   ...
   <!-- Document-level event binding -->
   <wml:onevent type="timer">
     <wml:go href="/general"/>
   </wml:onevent>

   <wml:card id="first">
     <!-- Document-level event binding not overridden. The event
          binding is active in this card. -->

     <!-- rest of card -->
     ...
   </wml:card>

   <wml:card id="second">
     <!-- Document-level event binding is overridden with noop.
          Binding is inactive in this card. -->
     <wml:onevent type="timer">
       <wml:noop/>
     </wml:onevent>
```

```
      <!-- rest of card -->
      ...
    </wml:card>

    <wml:card id="third">
      <!-- Document-level event binding is overridden. It is
           replaced by a card-level event binding. -->
      <wml:onevent type="timer">
        <wml:go href="/specific"/>
      </wml:onevent>

      <!-- rest of card -->
      ...
    </wml:card>
  </html>
```

5.7. Identification of Document Types

WML2 documents are identified by the MIME media type "application/wml+xml".

XHTML Mobile Profile documents are identified by the MIME media type “application/vnd.wap.xhtml+xml”.

XHTML Basic documents, however, do not have a unique MIME media type.

The type "application/xhtml+xml" can be used to identify documents from any of the XHTML-based markup languages, including XHTML Basic.

Section 2.1, Document Conformance, of [XHTMLBasic], gives a set of criteria that all conforming XHTML Basic documents must meet. A document that meets all these criteria is positively identified as an XHTML Basic document. However, as such criteria require validating the document according to the DTD, they are not useful for all user agents.

For user agents that do not validate documents according to the DTD, the following criteria may help to identify a document with media type "application/xhtml+xml" as an XHTML Basic document:

An XHTML Basic document must have the string "<!DOCTYPE html" preceding the root element, indicating a DOCTYPE declaration. (See 6.17 for details about document conformance).

The DOCTYPE declaration may include the XHTML Basic Formal Public Identifier and may also include the URI of the XHTML Basic DTD as specified below:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML Basic 1.0//EN"
    "http://www.w3.org/TR/xhtml-basic/xhtml-basic10.dtd">
```

However, the Formal Public Identifier is not required, and the system identifier may be modified appropriately.

There is no requirement that XHTML Basic documents be given the media type "application/xhtml+xml"; the media type "text/html" may be used instead. As there are no conformance rules for documents with type "text/html", there is no easy way to determine which documents of type "text/html" are XHTML Basic documents, except that the document may include the document type declaration specified above.

A conforming user agent MUST accept WML2 documents identified as "application/wml+xml".
A conforming user agent MUST accept XHTML Mobile Profile documents identified as “application/vnd.wap.xhtml+xml”. A conforming user agent SHOULD accept XHTML Mobile Profile documents identified as “application/xhtml+xml”.

5.8. Common User Agent Behaviour depending on the type of Elements

5.8.1. Activation of Elements using Access Keys

When the `accesskey` attribute or `-wap-accesskey` style sheet property is used, pressing the specified access key causes the element to be activated, or brought into focus [HTML4]. The result of activation varies by element. The following table defines activation for the elements of WML2.

Table 5-4. Activation for the elements of WML2

Elements	Activation action
`a`	Execute the associated task
`wml:anchor`	Execute the associated task
`input`	Give focus to the form control and allow input
`textarea`	Give focus to the form control and allow input
other elements that can be activated	Implementation dependent

5.9. The BACK Key in WML2

Unless otherwise overridden, the default behaviour of the BACK key is as follows:

- When the BACK key is activated by the end user, the user agent MUST execute a prev task as defined in section 5.3.2.

Through the use of the `wml:do` element, the author may override the default behaviour of the BACK key. When the author specifies a `wml:do` element with the pre-defined role of`prev`, the default behaviour of the WML2 user agent is overridden with the task specified by the `wml:do` element. See section 6.16.3 for details of the `wml:do` element.

5.10. Navigation User Interface Using the `wml:do` Element

The `wml:do` element provides a general mechanism for the user to act upon the current body or card (i.e., the current `body` element or `wml:card` element). A task is associated with the `wml:do` element and that task is executed when the user activates the element.

The presentation of the `wml:do` element is user agent dependent. The author must only assume that the `wml:do` element is mapped to a unique user interface widget that the user can activate. In addition, the author is required to specify the role of the associated task within the application, using the `type` attribute. The `type` attribute is used by the user agent in presentation of the `wml:do` element, and is discussed below.

When the user activates a `wml:do` element, the user agent MUST execute the associated task. The user agent determines what it means for a `wml:do` element to be activated, but it is assumed that the means of activation is appropriate to the type of user interface widget. For example, a key is activated when the end user presses (or releases) the key.

5.10.1. Processing the type Attribute

The type attribute specifies the role of the associated task within the application. The role is a logical action, such as acceptance of a choice or a request for help. It provides information to the user agent about the author's intended use of the wml:do element, to assist in mapping the task onto a physical user interface construct.

The user agent MUST use the type attribute on the wml:do element to provide a suitable mapping onto a physical user interface construct. Although the exact mapping is user agent dependent, the user agent MUST support a default presentation for all values of the type attribute.

Authors must not rely on the semantics or behaviour of an individual type attribute value, or on the mapping of a role to a particular physical construct.

All type attribute values are reserved, except for those marked as experimental or vendor-specific.

The user agent MUST accept any type value, but MAY treat any unrecognised type value as the equivalent of unknown.

Table 5-5. Predefined roles for the wml:do element.

Role	Description
accept	Positive acknowledgement. (e.g. OK, Yes, Accept, Done, Next)
prev	Backward history navigation.
help	Request for help. May be context sensitive.
reset	Clearing or resetting state.
options	Context-sensitive request for options or additional operations.
delete	Delete item or choice.
unknown	A generic do element.
X-*, x-*	Experimental types. This set is not reserved.
vnd.*, VND.*, and any combination of [Vv][Nn][Dd].*	Vendor-specific or user agent-specific types. This set is not reserved. Vendors should allocate names with the format VND.CO-TYPE, where CO is a company name abbreviation and TYPE is the widget type. This convention is the same as for MIME media types. See [RFC2045] for more information.

5.10.2. Overriding the BACK Key Using the wml:do Element

The wml:do element can be used to override the default behaviour for the BACK key, as defined in section 5.9. For a wml:do element with the type attribute equal to prev, if the default presentation is selected, the user agent MUST override the default behaviour of the BACK key with the task associated with the wml:do element. If there is more than one wml:do element with the type attribute equal to prev, then the first wml:do element in document order overrides the BACK key.

5.11.
5.11. Timer Processing

This section describes how the user agent processes a timer.

The timer SHALL be initialised and started at card or document entry and SHALL be stopped when the card or document is exited. Card or document entry is any task or user action that results in the `wml:card` element or `body` element being activated, for example, navigating into the card. Card or document exit is defined as the execution of any task.

The value of a timer SHALL decrement from the initial value, triggering the delivery of a `timer` event on transition from a value of one to zero. If the user has not exited the card at the time of timer expiration, a `timer` event SHALL be delivered to the card or the body.

The user agent SHALL interpret the timer timeout value in units of one-tenth (1/10) of a second. However, the actual resolution of the timer maintained by the user agent is implementation dependent. The interaction of the timer with the user agent's user interface and other time-based or asynchronous device functionality is also implementation dependent.

If the value of the timeout is not a positive integral number, the user agent SHALL ignore the `wml:timer` element. A timeout value of zero (0) SHALL disable the timer.

Invoking a refresh task is considered an exit. The task SHALL stop the timer, commit its value to the context, and update the user agent accordingly. Completion of the refresh task is considered an entry to the card. At that time, the timer SHALL restart.

Note: Variables may be used to simulate resuming a timer.

5.12. Acceptance of XHTML

The user agent MUST accept a valid XHTML Basic document.

The user agent MUST accept a valid XHTML Mobile Profile[XHTMLMP] document.

5.13. User Agent Conformance Rules

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

Note: In the following text, “must” in the original text in the reference is updated with “MUST”.

1. In order to be consistent with the XML 1.0 Recommendation [XML], 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 an XHTML document as generic [XML], it SHALL only recognise 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 recognise, it MUST continue to process the children of that element. If the content is text, the text MUST be presented to the user.

5. If a user agent encounters an attribute it does not recognise, it MUST ignore the entire attribute specification (i.e., the attribute and its value).

6. If a user agent encounters an attribute value it does not recognise, 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 has not 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, a user agent that encounters characters or character entity references that are recognised 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. White space is handled according to the following rules. The following characters are defined in [XML] as white space characters:

   - SPACE ( )
   - HORIZONTAL TABULATION (	)
   - CARRIAGE RETURN (
)
   - LINE FEED (
)

   The XML processor normalises different systems' line end codes into one single LINE FEED character, that is passed up to the application.

   The user agent MUST process white space characters in the data received from the XML processor as follows:

   - All white space surrounding block elements SHOULD be removed.
   - Comments are removed entirely and do not affect white space handling. One white space character on either side of a comment is treated as two white space characters.
   - When the 'xml:space' attribute is set to 'preserve', white space characters MUST be preserved and consequently LINE FEED characters within a block MUST NOT be converted.
   - When the 'xml:space' attribute is not set to 'preserve', then:
     - Leading and trailing white space inside a block element MUST be removed.
     - LINE FEED characters MUST be converted into one of the following characters: a SPACE character, a ZERO WIDTH SPACE character (), or no character (i.e. removed). The choice of the resulting character is user agent dependent and is conditioned by the script property of the characters preceding and following the LINE FEED character.
     - A sequence of white space characters without any LINE FEED characters MUST be reduced to a single SPACE character.
     - A sequence of white space characters with one or more LINE FEED characters MUST be reduced in the same way as a single LINE FEED character.

   White space in attribute values is processed according to [XML].

   Note (*informative*): In determining how to convert a LINE FEED character a user agent should consider the following cases, whereby the script of characters on either side of the LINE FEED determines the choice of the replacement. Characters of COMMON script (such as punctuation) are treated as the same as the script on the other side:

   1. If the characters preceding and following the LINE FEED character belong to a script in which the SPACE character is used as a word separator, the LINE FEED character should be converted into a SPACE character. Examples of such scripts are Latin, Greek, and Cyrillic.
   2. If the characters preceding and following the LINE FEED character belong to an ideographic-based script or writing system in which there is no word separator, the LINE FEED should be converted into no character. Examples of such scripts or writing systems are Chinese, Japanese.
   3. If the characters preceding and following the LINE FEED character belong to a non ideographic-based script in which there is no word separator, the LINE FEED should be converted into a ZERO WIDTH SPACE character () or no character. Examples of such scripts are Thai, Khmer.

4. If none of the conditions in (1) through (3) are true, the LINE FEED character should be converted into a SPACE character.

The Unicode [UNICODE] technical report TR#24 (Script Names) provides an assignment of script names to all characters.

6. WML2 Markup Elements and Attributes (Normative)

Interoperability is guaranteed when the authors observe the WML2 DTD. It is out of scope of this specification to guarantee interoperability when another DTD is supported by the user agent.

6.1. XHTML Basic and Extensions

WML2 markup elements and attributes consist of those elements and attributes from XHTML Basic [XHTMLBasic] with additional XHTML Modularization extensions and WAP extensions.

The Style Sheet and Presentation modules are defined in XHTML Modularization [XHTMLMod]. The Events and Context and Navigation Modules are WAP extensions. The WAP extensions in XHTML Modularization-based modules are defined in the following sections.

Except where specified in this document, the semantics of all the elements and attributes are defined in [HTML4].

The WML2 DTD is defined using XML Namespaces [XMLN]. The user agent MAY implement a fully namespaces-aware XML processor as defined by [XMLN], but is not required to do so in order to correctly process WML2 documents.

6.2. The Structure Module

The following elements in the Structure Module are used to specify the structure of the document:

```
body html wml:card head title
```

For the details of these elements, see [HTML4] unless stated otherwise in this specification.

6.2.1. The body Element

The `wml:newcontext` attribute specifies whether the browser context is initialised to a well-defined state when the document is loaded. If the `wml:newcontext` attribute value is “true”, the user agent MUST reinitialise the browser context upon navigation to this card.

6.2.2. The `html` Element

The `xmlns:wml` attribute refers to the WML namespace: `http://www.wapforum.org/2001/wml`.

The `wml:use-xml-fragments` attribute is used to specify how a fragment identifier is interpreted by the user agent. For details of use of `wml:use-xml-fragments` in the go task and the prev task, see sections 5.3.1 and 5.3.2.

6.2.3. The wml:card Element

The `wml:card` element specifies a fragment of the document body. Multiple `wml:card` elements may appear in a single document. Each `wml:card` element represents an individual presentation and/or interaction with the user.

If the `wml:card` element’s newcontext attribute value is “true”, the user agent MUST reinitialise the browser context upon navigation to this card.

6.3. Text Module

The following elements in the Text Module [XHTMLMod] are used to specify different types of text:

```
abbr acronym address blockquote br cite code dfn div em h1 h2 h3 h4 h5 h6
kbd p pre q samp span strong var
```

For the details of these elements, see [HTML4] unless stated otherwise in this specification.

6.3.1. The p Element

The following attributes have equivalents in the WCSS properties.

`align wml:mode`

The `align` attribute is equivalent to the `text-align` WCSS property. The `wml:mode` is equivalent to the `white-space` WCSS property, where the attribute value “wrap” is equal to the property value “normal” and the attribute value “nowrap” is equal to the property value “nowrap”. These styling attributes are deprecated and the authors are encouraged to use the WCSS style properties.

6.4. Hypertext Module

The following elements in the Hypertext Module [XHTMLMod] are used to define hypertext links to other resources:

a

For the a element, refer to [HTML4].

6.5. Forms Module

The following elements in the Forms Module are used to specify user interaction:

`form input select option label textarea optgroup fieldset`

For these elements, refer to [HTML4].

The `optgroup` and `fieldset` elements are part of the XHTML Forms Module, which extends the Basic Forms Module [XHTMLMod].

6.5.1. The `select` Element

The `wml:name` attribute specifies the name of the variable to set with the result of the selection. The `wml:value` attribute specifies the default-selected `option` element. The `wml:iname` attribute specifies the name of the variable to be set with the index of the selection. The `wml:ivalue` attribute specifies the index of the default-selected `option` element. For details of setting control variables, see section 5.4.3.2.

6.5.2. Text Control Format Attributes

6.5.2.1. The `wml:format` Attribute

The `wml:format` attribute specifies an input mask for user input entries. The *input mask* is a string consisting of mask control characters and static text that is displayed in the input area. The user agent may use the input mask to facilitate accelerated data input (e.g. the user agent may change its input mode according to the format code of the current position of the input cursor). An input mask is only valid when it contains legal format codes and static text. User agents MUST ignore invalid masks.

Character categories are as defined by [UNICODE]:

- “Letter” refers to character categories Lu, Ll, Lm, and Lo.
- “Uppercase letter” refers to character categories Lu and Lm.
- “Lowercase letter” refers to character categories Ll and Lm.
- “Numeric character” refers to the character category Nd.
- “Punctuation” refers to character categories Pc, Pd, Ps, Pe, and Po.
- “Symbol” refers to character categories Sm, Sc, Sk, and So.

User agents need only be capable of displaying and accepting the subsets of the above sets that are appropriate for all languages that they support. However, the user agent MUST support ASCII graphic characters of the Unicode Basic Latin block (U+0020 – U+007E).

For a given input element, the user agent may choose to restrict the set of allowable characters to those appropriate for the current language(s).

The *current languages* are the superset of:

- the current language of the WML document, plus
- the user agent's accept language(s), plus
- the user agent's interface language.

In caseless languages, format codes distinguishing between upper and lowercase are equivalent.

The format control characters specify the data format expected to be entered by the user.

The default format is "*M". The format codes are:

A entry of any uppercase letter, symbol, or punctuation character. Numeric characters are excluded.

a entry of any lowercase letter, symbol, or punctuation character. Numeric characters are excluded.

N entry of any numeric character.

n entry of any numeric, symbol, or punctuation character.

X entry of any uppercase letter, numeric character, symbol, or punctuation character.

x entry of any lowercase letter, numeric character, symbol, or punctuation character.

M entry of any character valid in the current languages, including any letter, numeric, symbol, or punctuation character. If the language supports case and the device supports both upper and lower case entry, the user agent may choose to default to uppercase entry mode but MUST allow entry of any character.

m entry of any character valid in the current languages, including any letter, numeric, symbol, or punctuation character. If the language supports case and the device supports both upper and lower case entry, the user agent may choose to default to lowercase entry mode but MUST allow entry of any character.

<lang:class> entry of any character belonging to language-specific class. A language-specific class character is one that delimited by a pair of '<' and '>'characters in the input format. The *lang* parameter is a language identifier as defined in [RFC3066]; i.e., a primary tag optionally followed by a subtag. The *class* parameter identifies a subset of characters specific to the identified language, and is vendor-specific or user agent specific parameter. If the user agent supports the language specified by *lang*, the user agent MUST restrict input to the characters which are members of the specified *class*. Otherwise (i.e., the user agent does not support the specified language or class), the user agent MUST treat this format code as "M".

****f*** entry of any number of characters; f is one of the above format codes and specifies what kind of characters can be entered. *Note: This format may only be specified once and must appear at the end of the format string.*

nf entry of up to *n* characters where *n* is natural number, f is one of the above format codes (other than **f* format code) and specifies what kind of characters can be entered. *Note: This format may only be specified once and must appear at the end of the format string.*

c display the character, *c*, in the entry field; allows escaping of the format codes as well as introducing non-formatting characters so they can be displayed in the entry area. Escaped characters are considered part of the input's value, and MUST be preserved by the user agent. For example, the stored value of the input "12345-123" having a mask "NNNNN\-3N" is "12345-123" and not "12345123". Similarly, if the value of the variable named by the name attribute is "12345123" and the mask is "NNNNN\-3N", the user agent MUST unset the variable since it does not conform to the mask.

6.5.2.2. The `wml:emptyok` Attribute

The `wml:emptyok` attribute indicates whether this input element accepts empty input or not. If `wml:emptyok` is "true", input is not required even if the format mask would otherwise require it. If `wml:emptyok` is "false", input is required even if the format mask would otherwise not require it.

If the author does not explicitly specify the `wml:emptyok` attribute, the format attribute fully defines the input requirement. The implied value of the `wml:emptyok` attribute is "true" when the `wml:format` attribute allows empty input (i.e., the format mask is implied or a "*f" format code). Otherwise, the implied value of the attribute is "false".

Whether or not input is required, any input given MUST match the format specification.

For the following input elements, input is required:

```
<input name="x" wml:format="M*M"/> <!-- implied: wml:emptyok="false" -->
<input name="x" wml:emptyok="false"/> <!-- implied: wml:format="*M" -->
<input name="x" wml:emptyok="false" wml:format="M*M"/>
<input name="x" wml:emptyok="false" wml:format="*M"/>
```

For the following input elements, input is not required:

```
<input name="x"/> <!-- implied: wml:format="*M" wml:emptyok="true" -->
<input name="x" wml:format="*M"/> <!-- implied: wml:emptyok="true" -->
<input name="x" wml:emptyok="true"/> <!-- implied: wml:format="*M" -->
<input name="x" wml:emptyok="true" wml:format="M*M"/>
<input name="x" wml:emptyok="true" wml:format="*M"/>
```

6.5.3. Form Control Initialisation Attribute

6.5.3.1. The `wml:name` Attribute

The `wml:name` attribute names the form control variable. For details about form control variables, see section 5.4.3.2.

6.5.4. Activation and Event Handler Attributes

6.5.4.1. The `accesskey` Attribute

The `accesskey` attribute names the key to activate the form control. See section 6.4 for a description.

6.5.4.2. The wml:onpick Attribute

The `wml:onpick` attribute in the `option` element specifies a URI to be navigated to when the `pick` event occurs.

The `pick` event occurs when the user selects or deselects this option. A multiple-selection option list generates a `pick` event whenever the user selects or deselects this option. A single-selection option list generates a `pick` event when the user selects this option, i.e., no event is generated for the de-selection of any previously selected option.

6.6. Tables Module

The following elements in the XHTML Tables Module are used to define tables:

```
caption table tr td th
```

For these elements, refer to [HTML4] unless otherwise specified in the following section.

6.6.1. The `table` Element

The `wml:align` attribute specifies the layout of text and images within the columns of a table. A column's contents can be centre aligned, left aligned or right aligned when it is rendered to the user. The attribute value is interpreted as a non-separated list of alignment designations, one for each column. Centre alignment is specified with the value "C", left alignment is specified with the value "L", right alignment is specified with the value "R", and default alignment is specified with the value "D". Designators are applied to columns as they are defined in the content. The first designator in the list applies to the first column, the second designator to the second column, and so forth. Default alignment is applied to columns that are missing alignment designators or have unrecognised designators. All extra designators are ignored. Determining the default alignment is implementation dependent. User agents SHOULD consider the current language when determining the default alignment and the direction of the table. A user agent MAY use other algorithms to make such decisions.

The `wml:align` attribute is deprecated. Authors are encouraged to use WCSS instead.

The presentation of the table is likely to depend on the display characteristics of the device. WML does not define how a user agent renders a table. User agents may create aligned columns for each table, or it may use a single set of aligned columns for all tables in a card. User agents that choose to render a table in a traditional tabular manner should determine the width of each column from the maximum width of the text and images in that column to ensure the narrowest display width. However, the user agent may use fixed width or other appropriate layout algorithms instead. The user agent that chooses to render tables in a traditional tabular manner MUST use a non-zero width gutter to separate each non-empty column.

6.7. Lists Module

The following elements in the XHTML Lists Module are used to define lists:

```
dt dd dl ol ul li
```

For these elements, refer to [HTML4].

The `start` attribute of the `ol` element and the `value` attribute of the `li` elements are part of the XHTML Legacy Module, which extends the Lists Module [XHTMLMod].

6.8. Image Module

The following element in the XHTML Image Module is used to provide basic image embedding:

```
img
```

For this element, refer to [HTML4] otherwise specified in the following section.

6.8.1. The `img` Element

The `wml:localsrc` attribute specifies an alternative internal representation for the image. This representation is used if it exists; otherwise the image is downloaded from the URI specified in the `src` attribute, i.e., any `wml:localsrc` parameter specified takes precedence over the image specified in the `src` parameter.

An author may use the `wml:localsrc` attribute to refer to a pictogram, as specified by [PICTO].

The following attributes have equivalents in the WCSS properties:

```
align vspace hspace
```

The `align` attribute is equivalent to the `vertical-align` property in the WCSS. The vspace and hspace attribute can be mapped to the `margin` property in the CSS. These styling attributes are deprecated and the authors are encouraged to use the WCSS style properties, if possible.

6.9. Metainformation Module

The following element in the XHTML Metainformation Module [XHTMLMod] is used to describe information within the declarative portion of a document:

```
meta
```

For this element, refer to [HTML4] otherwise specified in the following section.

6.9.1. The `meta` Element

For documents retrieved via HTTP, origin servers SHOULD use the property name specified by the `http-equiv` attribute to create an HTTP response header. For use of HTTP, see [WAE].

For a WML2 document, the user agent SHALL assume that CSS (text/css) is the default style sheet language if no "Content-Style-Type" reponse header is present and no `meta` element with the `http-equiv` attribute value of "Content-Style-Type" is specified.

The equivalent `meta` element would be written:

```
<meta http-equiv="Content-Style-Type" content="text/css"/>
```

If the WML document is processed by an intermediate agent, any `meta` element with `wml:forua` set to “false” SHOULD be removed before the document is sent to the client. If the `wml:forua` attribute is “true” the meta data of the element SHOULD be delivered to the user agent. The method of delivery may vary. For example, http-equiv meta-data may be delivered using HTTP or WSP headers.

A user agent SHOULD ignore any meta element with the `wml:forua` attribute set to “false”.

The `wml:forua` attribute is deprecated because WAP 2.0 Architecture cannot guarantee the semantics of this attribute.

6.10. Link Module

The following element in the XHTML Link Module [XHTMLMod] is used to define links to external resources:

```
link
```

For this element, refer to [HTML4].

The default value for `media` attribute is `all`.

6.11. Base Module

The following element in the XHTML Base Module [XHTMLMod] is used to define a base URI against which relative URIs in the document will be resolved:

```
base
```

For this element, refer to [HTML4].

6.12. Object Module

The following elements in the XHTML Object Module [XHTMLMod] are used to define general-purpose object inclusion:

```
object param
```

For these elements, refer to [HTML4].

6.13. Style Sheet Module

The following element in the XHTML Style Sheet Module [XHTMLMod] is used to declare an internal style sheet:

```
style
```

For this element, refer to [HTML4].

The default value for the `media` attribute is `all`.

6.14. Presentation Module

The following elements are used to define simple presentation-related markup:

```
b i small big u hr
```

These elements are not defined in XHTML Basic [XHTMLBasic]. All these elements except the `u` element are defined in the XHTML Presentation Module [XHTMLMod]. The `u` element is defined in the XHTML Legacy Module [XHTMLMod].

For these elements, refer to [HTML4].

The `b`, `i`, `small`, `big` and `u` elements are deprecated. Authors are encouraged to use the equivalent WCSS style properties for achieving the same functionalities.

6.15. Events Module

6.15.1. The `wml:onevent` Element

The `wml:onevent` element binds a task to a particular event for the immediately enclosing element, e.g., specifying a `wml:onevent` element inside a `wml:card` element associates an event binding with the `wml:card` element.

The user agent MUST ignore any `wml:onevent` element specifying a type that does not correspond to a legal event for the immediately enclosing element.

The `type` attribute indicates the name of the event. In this specification, only intrinsic event types are defined. See 5.6.1 for the types of intrinsic events.

6.16. Context and Navigation Module

6.16.1. The `wml:anchor` Element

The `wml:anchor` element defines a source anchor for a hyperlink that may be activated by the user. The source anchor is identical to a source anchor created with the `a` element, except that the destination anchor for the hyperlink is specified by the task contained within the `wml:anchor` element.

When the user activates the `wml:anchor` element, the user agent MUST execute the associated task.

Source anchors may be present in any text flow, excluding the text in `option` elements (i.e., anywhere formatted text is legal, except for `option` elements). It is an error to specify more than one task element (e.g., `wml:go`, `wml:prev` or `wml:refresh`) in an `wml:anchor` element.

The `accesskey` attribute assigns an access key to an element as defined in [HTML4]. The `accesskey` attribute is equivalent to the WCSS `wap-accesskey` property. The authors are encouraged to use the WCSS style properties, if possible.

6.16.2. The `wml:access` Element

The `wml:access` element specifies access control information for the entire document. It is an error for a document to contain more than one `wml:access` element. If a document does not include a `wml:access` element, access control SHALL be disabled. When access control is disabled, `wml:card` elements or `body` elements in any document can access this document.

A document's `domain` and `path` attributes specify which other documents may access it. As the user agent navigates from one document to another, it performs access control checks to determine whether the destination document allows access from the current document.

If a document has a `domain` and/or `path` attribute, the user agent MUST verify that the referring document's URI matches the values of the attributes. Matching is done as follows: the access domain is suffix-matched against the domain name portion of the referring URI and the access path is prefix matched against the path portion of the referring URI. When the verification fails, the user agent MUST deny access.

The user agent MUST perform domain suffix matching using the entire element of each sub-domain and verify an exact match of each element (e.g., `www.wapforum.org` shall match `wapforum.org`, but shall not match `forum.org`).

The user agent MUST perform path prefix matching using entire path elements and verify an exact match of each element (e.g., `/X/Y` matches `path="/X"` attribute, but does not match `path="/XZ"` attribute). When the verification fails, the user agent MUST deny access.

The domain attribute defaults to the current document's domain. The path attribute defaults to the value "/".

To simplify the development of applications that may not know the absolute path to the current document, the `path` attribute accepts relative URIs. The user agent converts the relative path to an absolute path and then performs prefix matching against the `path` attribute.

For example, given the following access control attributes:

```
domain="wapforum.org"
path="/cbb"
```

The following referring URIs would be allowed to go to the document:

```
http://wapforum.org/cbb/stocks.cgi
https://www.wapforum.org/cbb/bonds.cgi
http://www.wapforum.org/cbb/demos/alpha/packages.cgi?x=123&y=456
```

The following referring URIs would not be allowed to go to the document:

```
http://www.test.net/cbb
http://www.wapforum.org/internal/foo.wml
```

The `domain` and `path` attributes follow URI case sensitivity rules [RFC2396].

6.16.3. The `wml:do` Element

The `wml:do` element provides a general mechanism for the user to act upon the current body/card (i.e., the current `body` element or `wml:card` element). A task is associated with the `wml:do` element and that task is executed when the user activates the element.

The `type` attribute specifies the role of the associated task within the application. For the details of type processing, see section 5.10.1

If the user agent is able to dynamically label the user interface widget, the `label` attribute specifies a textual string suitable for such labelling. The user agent SHOULD make a best-effort attempt to label the UI widget and SHOULD adapt the label to the constraints of the widget (e.g., truncate the string). If an element can not be dynamically labelled, this attribute may be ignored.

To work well on a variety of user agents, labels should be six characters or shorter in length.

6.16.4. The `wml:go` Element

The `href` attribute specifies a destination URI.

The `sendreferer` attribute specifies the control of the referring document information. If the `sendreferer` attribute is true, the user agent MUST specify, for the server's benefit, the URI of the document containing this task (i.e., the referring document).

The `method` attribute specifies the HTTP submission method.

The `enctype` attribute specifies the media type used to submit the parameter to the server.

```
application/x-www-form-urlencoded
multipart/form-data
application/vnd.wap.wml.form.urlencoded
```

Note: `application/vnd.wap.wml.form.urlencoded` is added for backwards compatibility.

The accept-charset attribute specifies the list of character encodings for data that the origin server must accept when processing input. The value of this attribute is a comma- or space-separated list of character encoding names as specified in [RFC2045] and [RFC2616].

If the accept-charset attribute is not specified or is the reserved string unknown, the user agent should use the character encoding that was used to transmit the document that contains the wml:go element.

The cache-control attribute specifies the control of cache use. If the cache-control attribute is present, and the value is set to "no-cache", the client MUST reload the URI from the origin server. This attribute represents the HTTP "cache-control" header. When this attribute is present, the HTTP "cache-control" header MUST be added to the request with the same value as specified in the attribute.

The type attribute gives an advisory hint as to the media type of the content available at the link target address. It allows user agents to opt to use a fallback mechanism rather than fetch the content if they are advised that they will get content in a media type they do not support.

Authors who use this attribute take responsibility to manage the risk that it may become inconsistent with the content available at the link target address.

The wml:go element declares a go task, indicating navigation to a URI.

If the wml:go element's href attribute names a WML card or document, the user agent SHALL display the destination card or document.

A wml:go task SHALL execute a "push" operation on the history stack.

The user agent SHALL ignore all wml:postfield elements in a wml:go element if the target of the wml:go element is a card contained within the current document.

The wml:go element may contain one or more wml:postfield elements. These elements specify information to be submitted to the origin server during the request. The submission of field data is performed in the following manner:

1. The field name/value pairs are identified and all variables are substituted.
2. The user agent SHOULD transcode the field names and values to the correct character set, as specified explicitly by the accept-charset or implicitly by the document encoding.
3. If the href attribute value is an HTTP URI, the request is performed according to the method and enctype attributes' values:

Table 6-1. Processing according to the method and enctype attributes' values

Method	Enctype	Process
get	application/vnd.wap.wml.form.urlencoded	The field names and values are escaped using URI-escaping and assembled into an application/x-www-form-urlencoded media type. When a wml:postfield element is used to name a field, and the value of the wml:postfield's value attribute references the control variable for a multiple-selection control, the field value MUST be submitted as a semicolon-separated list of values. For example, for a field named "x" with value "$(x) ", where "x" is the control variable with value "choice1;choice3", the submitted data will be "x=choice1;choice3". The submission data is appended to the query component of the URI. The result MUST be a valid query component with the original query part and the postfields combined. An HTTP GET operation is performed on the resulting URI.

Method	Enctype	Process
	`application/x-www-form-urlencoded`	The field names and values are escaped using URI-escaping and assembled into an `application/x-www-form-urlencoded` media type. When a `wml:postfield` element is used to name a field, and the value of the `wml:postfield`'s `value` attribute references the control variable for a multiple-selection control, the field value MUST be submitted as specified in [HTML4], where one name/value pair is included for each item in the list. For example, for a field named "x" with value "$(x) ", where "x" is the control variable with value "choice1;choice3", the submitted data will be "x=choice1&x=choice3". The submission data is appended to the query component of the URI. The result MUST be a valid query component with the original query part and the postfields combined. An HTTP GET operation is performed on the resulting URI.
	`multipart/form-data`	Error. The user agent MUST ignore the `wml:go` element.
`post`	`application/vnd.wap.wml.form.urlencoded`	The field names and values are escaped using URI-escaping and assembled into an `application/x-www-form-urlencoded` media type. When a `wml:postfield` element is used to name a field, and the value of the `wml:postfield`'s `value` attribute references the control variable for a multiple-selection control, the field value MUST be submitted as a semicolon-separated list of values. For example, for a field named "x" with value "$(x) ", where "x" is the control variable with value "choice1;choice3", the submitted data will be "x=choice1;choice3". The submission data is sent as the body of the HTTP POST request. The `Content-Type` header MUST include the `charset` parameter to indicate the character encoding.

Method	Enctype	Process
	`application/x-www-form-urlencoded`	The field names and values are escaped using URI-escaping and assembled into an `application/x-www-form-urlencoded` media type. When a `wml:postfield` element is used to name a field, and the value of the `wml:postfield`'s `value` attribute references the control variable for a multiple-selection control, the field value MUST be submitted as specified in [HTML4], where one name/value pair is included for each item in the list. For example, for a field named "x" with value "$(x) ", where "x" is the control variable with value "choice1;choice3", the submitted data will be "x=choice1&x=choice3". The submission data is sent as the body of the HTTP POST request. The `Content-Type` header MUST include the `charset` parameter to indicate the character encoding when the part contains characters not in the US-ASCII character set.

	`multipart/form-data`	The field names and values are encoded as a `multipart/form-data` media type as defined in [RFC2388].
		
		The submission data is sent as the body of the HTTP POST request.
		
		The `Content-Type` header MUST include the `charset` parameter to indicate the character encoding when the part contains characters not in the US-ASCII character set.
		
		When a `wml:postfield` element is used to name a field, and the value of the `wml:postfield`'s `value` attribute references the control variable for a multiple-selection control, the field value MUST be submitted as specified in [HTML4], where one part is included for each item in the list.
		
		For example, for a field named "x" with value "$(x) ", where "x" is the control variable with value "choice1;choice3", the submitted data will be:
		
		`Content-Disposition: form-data; name="x"`
		
		`choice1`
		
		`----------AaAaAax`
		
		`Content-Disposition: form-data; name="x"`
		
		`choice3`

6.16.5. The `wml:noop` Element

The `wml:noop` element specifies that nothing should be done, i.e., "no operation". This task is used in conjunction with the `wml:onevent` element to make inactive an event handler task that would otherwise be active.

6.16.6. The wml:prev Element

The wml:prev element declares a prev task, indicating navigation to the previous URI on the history stack. The prev task performs a “pop” operation on the history stack. See section 5.3.2 for processing of the prev task.

6.16.7. The wml:refresh Element

The wml:refresh element declares a refresh task, indicating an update of the user agent context as specified by the wml:setvar elements. See section 5.3.4 for processing of the refresh task.

6.16.8. The wml:postfield Element

The name attribute specifies the field name. The value attribute specifies the field value.

The wml:postfield element is used to indicate an instance of a variable or program argument. The wml:postfield element specifies a field name and value for transmission to an origin server during a URI request.

The actual encoding of the name and value will depend on the method used to communicate to the origin server.

6.16.9. The wml:setvar Element

The name attribute specifies the name of the variable. The value attribute specifies the expression to evaluate and assign the variable.

The wml:setvar element specifies the variable to set in the current browser context as a side effect of executing a task.

The user agent SHALL ignore the element if the name attribute does not evaluate to a legal variable name at runtime.

The user agent SHALL ignore the element if the value attribute does not evaluate to a legal expression at runtime.

6.16.10. The wml:getvar Element

The wml:getvar element is used to substitute the value of a variable into the text (#PCDATA) of a document.

The wml:getvar element is used in conjunction with the attribute expression syntax, which allows authors to insert variable references into attribute values. Together, they provide a mechanism by which WML documents can be parameterised. For the details of the attribute expression syntax, see section 5.5.

The required name attribute specifies the name of the variable whose value is the substitution text. If the variable is not set, or if the name attribute value does not evaluate to a legal variable name, the substitution text is the empty string. The user agent MUST replace the element with the substitution text.

The conversion attribute specifies the conversion to be applied to the value of the variable. The conversions involve URI escaping and unescaping, and are identical to those specified for variable references in section 5.5.1.1. Legal values are “noesc”, “escape” and “unesc”.

Only textual information can be substituted; no substitution of elements or attributes is possible. The user agent MUST NOT attempt to process the resulting string as XML markup.

The use of the wml:getvar element is illustrated by the following example.

Given a variable “x” with a value “100.00”, the following markup can be used to substitute the value of “x” into a paragraph:

```
<p>Your balance is $<wml:getvar name="x"/>.</p>
```

This markup would be displayed as:

```
Your balance is $100.00.
```

6.16.11. The `wml:timer` Element

The `wml:timer` element declares a timer for a card or body, which exposes a means of processing inactivity or idle time.

The `name` attribute specifies the name of the variable to be set with the value of the timer. The `name` variable's value is used to set the timeout period upon timer initialisation. The variable named by the `name` attribute will be set with the current timer value when the card is exited or when the timer expires. For example, if the timer expires, the `name` variable is set to a value of "`0`".

The `value` attribute indicates the default value of the variable named in the `name` attribute. When the timer is initialised and the variable named in the `name` attribute is not set, the `name` variable is assigned the value specified in the `value` attribute. If the `name` variable already contains a value, the `value` attribute is ignored. If the `name` attribute is not specified, the timeout is always initialised to the value specified in the `value` attribute.

For the details of timer processing, see section 5.11.

6.17. Document Conformance

A conforming WML2 document MUST meet all of the following criteria:

- The document MUST conform to the constraints expressed in Appendix A.
- The root element of the document MUST be <html>.
- The name of the default namespace on the root element MUST be the XHTML namespace name, `http://www.w3.org/1999/xhtml`. Also on the root element, the WML namespace MUST be declared with the name, `http://www.wapforum.org/2001/wml, and the prefix, "wml".`

  ```
  <html
  xmlns="http://www.w3.org/1999/xhtml"
  xmlns:wml="http://www.wapforum.org/2001/wml">
  ```
- There MUST be a DOCTYPE declaration with a public identifier in the document prior to the root element. The public identifier included in the DOCTYPE declaration must reference the DTD found in Appendix A using its Formal Public Identifier. The system identifier may be modified appropriately.

  ```
  <!DOCTYPE html PUBLIC"-//WAPFORUM//DTD WML 2.0//EN"
  "http://www.wapforum.org/DTD/wml20.dtd">
  ```
- The DTD subset MUST NOT be used to override any parameter entities in the DTD.

7. Use of Style Sheets with WML2 (Normative)

Style sheets can be used to style WML2 documents. If the user agent supports styling of documents with style sheets, it MUST support the style language WAP CSS [WCSS], a subset of CSS2 with WAP-specific extensions. A user agent MAY support other style languages.

7.1. Adding Style to WML2 Documents

Style information can be associated with a document in 3 ways:

- External style sheet
- Internal style information
- Inline style information

External Style Sheets

An external style sheet can be associated with a document using a special XML processing instruction or the `link` element. The use of the XML processing instruction is specified in [WCSS].

In the following example, the XML processing instruction is used to associate the external style sheet “mobile.css”:

```
<?xml-stylesheet href="mobile.css" media="handheld" type="text/css" ?>
```

The use of the `link` element (section 6.10) is specified by [HTML4].

To link an external style sheet to a document using the `link` element, `ref="stylesheet"` or `ref="alternate stylesheet"` is specified. In either case, the `type` attribute specifies the style sheet language.

For `type="text/css"`, the user agent MUST process the style sheet according to the style language WAP CSS [WCSS].

In the following example, the `link` element is used to associate the external style sheet “mystyle.css”:

```
<html>
  <head>
    <link href="mystyle.css" type="text/css" rel="stylesheet"/>
    ...
  </head>
  ...
</html>
```

Internal Style Sheets

Style information can be located within the document using the `style` element. This element, like `link`, must be located in the document header.

The following shows an example of an internal style sheet:

```
<html>
  <head>
    <style type="text/css">
    p { text-align: center; }
    </style>
    ...
  </head>
  ...
</html>
```

User agents that don't support style sheets, or don't support the specific style sheet language used by a `style` element, MUST hide the content of the `style` element.

Inline Style

An author can specify style information for a single element using the `style` attribute. This is called inline style. The `style` attribute is part of the Core attribute set and is therefore available on every element in WML2. The default style language for style information in the `style` attribute is WAP CSS.

In the following example, inline styling information is applied to a specific paragraph element:

```
<p style="text-align: center">...</p>
```

Note that not all styling rules apply to all elements, and some elements are completely unaffected by styling rules. See [WCSS] for details.

7.2. The Default Style Sheet for WML2

See Appendix B for the WML2 default style sheet.

Appendix A. The DTD for WML2 (Normative)

See http://www.wapforum.org/DTD/wml20.zip for the zipped file including DTDs for WML2.

It includes a flat version of the DTD, "wml20-flat.dtd". It includes a DTD driver and WML specific modules, "wml20.dtd" and "wml-*.mod".

See http://www.wapforum.org/DTD/wml20-flat.dtd for the flat version DTD for WML2.

See http://www.wapforum.org/DTD/wml20.dtd for the DTD driver for WML2.

Note that since we exclude all the W3C XHTML modules, in order to use the DTD driver, the user must download the W3C modules from the W3C Web site.

Appendix B. The WML2 Default Style Sheet (Informative)

```
/*  A sample style sheet for WML 2.0

    Modified from the HTML 4 CSS stylesheet found in the CSS 2
specification.
*/

body, card, div, p, center, hr, h1, h2, h3, h4, h5, h6,
address, blockquote, pre, ol, ul, dl, dt, dd,
form, fieldset, object
                    { display: block }
li              { display: list-item }
head            { display: none }
table           { display: table }
tr              { display: table-row }
td, th          { display: table-cell }
caption         { display: table-caption }
th              { font-weight: bolder; text-align: center }
caption         { text-align: center }
h1, h2, h3, h4, h5, h6, b,
strong          { font-weight: bolder }
i, cite, em, var,
address         { font-style: italic }
pre, code, kbd,
pre             { white-space: pre }
big                 { font-size: larger}
small               { font-size: smaller}
hr              { border: 1px inset }
ol              { list-style-type: decimal }
u                   { text-decoration: underline }
/* end wml-default.css */
```

Appendix C. The WML2 Elements (Informative)

Note: All the REQUIRED attributes are in bold. The default attribute value of an attribute, if one exists, is in italic.

Element	Attributes	Content
a	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), href(CDATA), charset(CDATA), type(CDATA), hreflang(NMTOKEN), rel(NMTOKENS), rev(NMTOKENS), accesskey(CDATA), tabindex(CDATA)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | img | object | wml:getvar | input | select | textarea | label)*
abbr	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
acronym	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
address	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
b	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
base	**href**(CDATA)	EMPTY
big	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar |

		input | select | textarea | label)*
blockquote	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), cite(CDATA)	(h1 | h2 | h3 | h4 | h5 | h6 | ul | ol | dl | p | div | pre | blockquote | address | hr | table | form | fieldset | wml:do)+
body	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), wml:newcontext('true' | '*false*'), wml:onenterforward(CDATA), wml:onenterbackward(CDATA), wml:ontimer(CDATA)	(wml:onevent*, wml:timer?, (h1 | h2 | h3 | h4 | h5 | h6 | ul | ol | dl | p | div | pre | blockquote | address | hr | table | form | fieldset | wml:do)+)
br	id(ID), class(NMTOKENS), title(CDATA), style(CDATA)	EMPTY
caption	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
cite	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
code	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
dd	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | h1 | h2 | h3 | h4 | h5 | h6 | ul | ol | dl | p | div | pre | blockquote | address | hr | table | form | fieldset | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*

dfn	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
div	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | h1 | h2 | h3 | h4 | h5 | h6 | ul | ol | dl | p | div | pre | blockquote | address | hr | table | form | fieldset | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
dl	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(dt | dd)+
dt	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
em	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
fieldset	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | h1 | h2 | h3 | h4 | h5 | h6 | ul | ol | dl | p | div | pre | blockquote | address | hr | table | form | fieldset | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
form	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), **action**(CDATA), method('*get*' | 'post'), enctype(CDATA '*application/x-www-form-urlencoded*')	(h1 | h2 | h3 | h4 | h5 | h6 | ul | ol | dl | p | div | pre | blockquote | address | hr |

		table | fieldset)+
h1	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
h2	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
h3	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
h4	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
h5	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
h6	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
head	xml:lang(NMTOKEN), profile(CDATA)	((meta | link | style | object | wml:access)*, ((title, (meta

		| link | style | object | wml:access)*, (base, (meta | link | style | object | wml:access)*)?) | (base, (meta | link | style | object | wml:access)*, (title, (meta | link | style | object | wml:access)*))))
hr	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	EMPTY
html	version('-//WAPFORUM//DTD WML 2.0//EN'), xml:lang(NMTOKEN), xmlns:wml('http://www.wapforum.org/2001/wml'), wml:onenterforward(CDATA), wml:onenterbackward(CDATA), wml:ontimer(CDATA), wml:use-xml-fragments('*true*' | 'false')	(head, (wml:onevent)*, (body | (wml:card+)))
i	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
img	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), **src**(CDATA), **alt**(CDATA), longdesc(CDATA), height(CDATA), width(CDATA), wml:localsrc(CDATA), wml:type(CDATA), vspace(CDATA '*0*'), hspace(CDATA '*0*'), align('top' | 'middle' | '*bottom*')	EMPTY
input	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), type('*text*' | 'password' | 'checkbox' | 'radio' | 'submit' | 'reset' | 'hidden'), name(CDATA), value(CDATA), checked('checked'), size(CDATA), maxlength(CDATA), src(CDATA), tabindex(CDATA), accesskey(CDATA), wml:format(CDATA), wml:emptyok('true' | 'false'), wml:name(CDATA)	EMPTY
kbd	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
label	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), for(IDREF), accesskey(CDATA)	(PCDATA | input | select | textarea | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | img | object | wml:getvar)*

li	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), value(CDATA)	(PCDATA | h1 | h2 | h3 | h4 | h5 | h6 | ul | ol | dl | p | div | pre | blockquote | address | hr | table | form | fieldset | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
link	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), charset(CDATA), href(CDATA), hreflang(NMTOKEN), type(CDATA), rel(NMTOKENS), rev(NMTOKENS), media(CDATA)	EMPTY
meta	xml:lang(NMTOKEN), http-equiv(NMTOKEN), name(NMTOKEN), **content**(CDATA), scheme(CDATA), wml:forua('*true*' | 'false')	EMPTY
object	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), declare('declare'), classid(CDATA), codebase(CDATA), data(CDATA), type(CDATA), codetype(CDATA), archive(CDATA), standby(CDATA), height(CDATA), width(CDATA), name(CDATA), tabindex(CDATA)	(PCDATA | h1 | h2 | h3 | h4 | h5 | h6 | ul | ol | dl | p | div | pre | blockquote | address | hr | table | form | fieldset | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label | param)*
ol	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), start(CDATA)	(li)+
optgroup	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), **label**(CDATA)	(option)+
option	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), selected('selected'), value(CDATA), wml:onpick(CDATA)	(PCDATA | wml:onevent | wml:getvar)*
p	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), wml:mode('wrap' | 'nowrap'), align('*left*' | 'right' | 'center')	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label | table | fieldset)*
param	id(ID), **name**(CDATA), value(CDATA), valuetype('*data*' | 'ref' | 'object'), type(CDATA)	EMPTY
pre	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), xml:space('preserve')	(PCDATA | br | span | i | b | em | strong | dfn | code | samp | kbd | var | cite | abbr |

		acronym | q | a | wml:anchor | wml:do | wml:getvar | input | select)*
q	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), cite(CDATA)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
samp	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
select	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), name(CDATA), size(CDATA), multiple('multiple'), tabindex(CDATA), wml:iname(NMTOKEN), wml:value(CDATA), wml:ivalue(CDATA), wml:name(CDATA)	(optgroup | option)+
small	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
span	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
strong	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
style	title(CDATA), xml:lang(NMTOKEN), **type**(CDATA), media(CDATA), xml:space('preserve')	PCDATA

table	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), summary(CDATA), wml:columns(CDATA), wml:align(CDATA)	(caption?, tr+)
td	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), abbr(CDATA), axis(CDATA), headers(IDREFS), scope('row' | 'col'), rowspan(CDATA '*1*'), colspan(CDATA '*1*'), align('left' | 'center' | 'right'), valign('top' | 'middle' | 'bottom')	(PCDATA | h1 | h2 | h3 | h4 | h5 | h6 | ul | ol | dl | p | div | pre | blockquote | address | form | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
textarea	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), name(CDATA), **rows**(CDATA), **cols**(CDATA), tabindex(CDATA), accesskey(CDATA), wml:format(CDATA), wml:emptyok('true' | 'false'), wml:name(CDATA)	(PCDATA | wml:getvar)*
th	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), abbr(CDATA), axis(CDATA), headers(IDREFS), scope('row' | 'col'), rowspan(CDATA '*1*'), colspan(CDATA '*1*'), align('left' | 'center' | 'right'), valign('top' | 'middle' | 'bottom')	(PCDATA | h1 | h2 | h3 | h4 | h5 | h6 | ul | ol | dl | p | div | pre | blockquote | address | form | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
title	xml:lang(NMTOKEN)	PCDATA
tr	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), align('left' | 'center' | 'right'), valign('top' | 'middle' | 'bottom')	(th | td)+
u	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label)*
ul	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(li)+
var	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | a | wml:anchor | wml:do | img | object | wml:getvar | input | select | textarea | label

		)*
wml:access	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), domain(CDATA), path(CDATA)	EMPTY
wml:anchor	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), accesskey(CDATA)	(PCDATA | br | span | em | strong | dfn | code | samp | kbd | var | cite | abbr | acronym | q | i | b | big | small | u | img | object | wml:getvar | input | select | textarea | label | wml:go | wml:prev | wml:refresh)*
wml:card	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), newcontext('true' | '*false*'), onenterforward(CDATA), onenterbackward(CDATA), ontimer(CDATA)	(wml:onevent*, wml:timer?, (h1 | h2 | h3 | h4 | h5 | h6 | ul | ol | dl | p | div | pre | blockquote | address | hr | table | form | fieldset | wml:do)+)
wml:do	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), xml:lang(NMTOKEN), **type**(CDATA), label(CDATA)	(wml:go | wml:prev | wml:noop | wml:refresh)
wml:getvar	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), **name**(CDATA), conversion('escape' | '*noesc*' | 'unesc')	EMPTY
wml:go	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), **href**(CDATA), sendreferer('true' | '*false*'), type(CDATA), method('post' | '*get*'), enctype(CDATA '*application/x-www-form-urlencoded*'), accept-charset(CDATA), cache-control('no-cache')	(wml:postfield | wml:setvar)*
wml:noop	id(ID), class(NMTOKENS), title(CDATA), style(CDATA)	EMPTY
wml:onevent	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), **type**(CDATA)	(wml:go | wml:prev | wml:noop | wml:refresh)
wml:postfield	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), **name**(CDATA), **value**(CDATA)	EMPTY
wml:prev	id(ID), class(NMTOKENS), title(CDATA), style(CDATA)	(wml:setvar)*
wml:refresh	id(ID), class(NMTOKENS), title(CDATA), style(CDATA)	(wml:setvar)*
wml:setvar	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), **name**(CDATA), **value**(CDATA)	EMPTY
wml:timer	id(ID), class(NMTOKENS), title(CDATA), style(CDATA), name(NMTOKEN), **value**(CDATA)	EMPTY

Appendix D. Static Conformance Requirements (Normative)

The notation used in this appendix is specified in [CREQ].

User Agent Context

Item	Function	Reference	Status	Requirement
WML-AGENTCONTEXT-C-001	WML variable context	5.2.1	M	
WML-AGENTCONTEXT-C-002	A navigation history	5.2.2	M	

Navigation Reference Processing Model

Item	Function	Reference	Status	Requirement
WML-NAVREF-C-001	The go task	5.3.1	M	
WML-NAVREF-C-002	The prev task	5.3.2	M	
WML-NAVREF-C-003	The refresh task	5.3.4	M	
WML-NAVREF-C-004	Task execution failure	5.3.5	M	

Form Processing Model

Item	Function	Reference	Status	Requirement
WML-FORMREF-C-001	Text input control initialisation	5.4.2.2	M	
WML-FORMREF-C-002	Menu control initialisation	5.4.2.3	M	
WML-FORMREF-C-003	Checkbox control initialisation	5.4.2.4	M	
WML-FORMREF-C-004	Radio button control initialisation	5.4.2.5	M	
WML-FORMREF-C-005	Submit button control initialisation	5.4.2.6	M	
WML-FORMREF-C-006	Hidden control initialisation	5.4.2.8	M	
WML-FORMREF-C-007	Text input control interaction	5.4.3.2	M	
WML-FORMREF-C-008	Menu input control Interaction	5.4.3.3	M	
WML-FORMREF-C-009	Checkbox button control interaction	5.4.3.4	M	
WML-FORMREF-C-010	Radio button control interaction	5.4.3.5	M	
WML-FORMREF-C-011	Submit button control interaction	5.4.3.6	M	
WML-FORMREF-C-012	Reset button control interaction	5.4.3.7	M	

Item	Function	Reference	Status	Requirement
WML-FORMREF-C-013	Committing form data	5.4.4	M	
WML-FORMREF-C-014	Form submission	5.4.5	M	

Attribute Expression Syntax

Item	Function	Reference	Status	Requirement
WML-ATTREXPR-C-001	Attribute expression syntax for WML2	5.5	M	
WML-ATTREXPR-C-002	No attribute expression syntax for XHTML Mobile Profile	5.5	M	
WML-ATTREXPR-C-003	Variable reference	5.5.1.1	M	
WML-ATTREXPR-C-004	Parsing the variable reference syntax	5.5.1.2	M	
WML-ATTREXPR-C-005	Variable scoping	5.5.1.4	M	
WML-ATTREXPR-C-006	Invalid variable references	5.5.1.5	M	
WML-ATTREXPR-C-007	Event parameters	5.5.1.6	M	
WML-ATTREXPR-C-008	Re-generation of presentation document	5.5.1.7	M	

Event Model

Item	Function	Reference	Status	Requirement
WML-EVENTREF-C-001	Event binding	5.6.3.1	M	
WML-EVENTREF-C-002	Bindings for untyped events	5.6.3.2	M	
WML-EVENTREF-C-003	Event binding scope	5.6.3.3	M	

Identification of Document Types

Item	Function	Reference	Status	Requirement
WML-DOCTYPES-C-001	XHTML Mobile Profile document type	5.7	M	
WML-DOCTYPES-C-002	WML2 document type	5.7	M	

BACK key

Item	Function	Reference	Status	Requirement
WML-BACKKEY-C-001	BACK key	5.9	M	

Timer Processing

Item	Function	Reference	Status	Requirement
WML-TIMERREF-C-001	Timer processing	5.11	M	

Acceptance of XHTML

Item	Function	Reference	Status	Requirement
WML-XHTMLBASIC-C-001	XHTML Basic	5.12	M	
WML-XHTMLBASIC-C-002	XHTML Mobile Profile	5.12	M	XHTMLMP:MCF

User Agent Conformance Rules

Item	Function	Reference	Status	Requirement
WML-AGENTCONF-C-001	XHTML User Agent Conformance Rules	5.13	M	

Structure Module

Item	Function	Reference	Status	Requirement
WML-STRUCTMOD-C-001	Initialisation in `wml:newcontext`	6.2.1	M	
WML-STRUCTMOD-C-002	Initialisation in `newcontext`	6.2.3	M	

Forms Module

Item	Function	Reference	Status	Requirement
WML-FORMSMOD-C-001	Mask control in `wml:format`	6.5.2.1	M	
WML-FORMSMOD-C-002	Input control in `wml:emptyok`	6.5.2.2	M	

Tables Module

Item	Function	Reference	Status	Requirement
WML-TABLESMOD-C-001	Column control in `wml:columns`	6.6.1	M	
WML-TABLESMOD-C-002	Column alignment using `wml:align`	6.6.1	O	

Image Module

Item	Function	Reference	Status	Requirement
WML-IMAGEMOD-C-	Image control in	6.8.1	M	

Item	Function	Reference	Status	Requirement
001	`wml:localsrc`			

Metainformation Module

Item	Function	Reference	Status	Requirement
WML-METAMOD-C-001	CSS as the default style sheet language	6.9.1	M	
WML-METAMOD-S-001	Processing the meta element with a wml:forua attribute	6.9.1	O	

Events Module

Item	Function	Reference	Status	Requirement
WML-EVENTSMOD-C-001	Binding in `wml:onevent`	6.15.1	M	

Context and Navigation Module

Item	Function	Reference	Status	Requirement
WML-CONTEXTMOD-C-001	Task activation in `wml:anchor`	6.16.1	M	
WML-CONTEXTMOD-C-002	Access control in `wml:access`	6.16.2	M	
WML-CONTEXTMOD-C-003	Task activation in `wml:go`	6.16.4	M	
WML-CONTEXTMOD-C-004	No operation in `wml:noop`	6.16.5	M	
WML-CONTEXTMOD-C-005	Backward navigation with `wml:prev`	6.16.6	M	
WML-CONTEXTMOD-C-006	Display update with `wml:refresh`	6.16.7	M	
WML-CONTEXTMOD-C-007	Submission of data with `wml:postfield`	6.16.8	M	
WML-CONTEXTMOD-C-008	Variable assignment in `wml:setvar`	6.16.9	M	
WML-CONTEXTMOD-C-09	Variable reference in `wml:getvar`	6.16.10	M	
WML-CONTEXTMOD-C-010	Timer control in `wml:timer`	6.16.11	M	

Use of Style Sheets

Item	Function	Reference	Status	Requirement
WML-USESTYLE-C-001	Support for WAP CSS	7	O	WCSS:MCF AND WML-USESTYLE-C-002 AND WML-USESTYLE-C-003 AND WML-USESTYLE-C-004
WML-USESTYLE-C-002	Handling of type “text/css” for external style sheet	7.1	O	
WML-USESTYLE-C-003	Handling of type “text/css” for internal style sheet	7.1	O	
WML-USESTYLE-C-004	Default type “text/css” for inline style rules	7.1	O	

Appendix E. Change History (Informative)

Type of Change	Date	Section	Description
Class 0	11-Sep-2001		The initial version of this document.