WAP WML

Version 30-Apr-1998

Wireless Application Protocol
Wireless Markup Language Specification

Disclaimer:

This document is subject to change without notice.

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 "*Wireless Application Protocol Architecture Specification"* [WAP].

This specification defines the Wireless Markup Language (WML). WML is a markup language based on [XML] and is intended for use in specifying content and user interface for narrowband devices, including cellular phones and pagers.

WML is designed with the constraints of small narrowband devices in mind. These constraints include:

- Small display and limited user input facilities
- Narrowband network connection
- Limited memory and computational resources

WML includes four major functional areas:

- Text presentation and layout - WML includes text and image support, including a variety of formatting and layout commands. For example, boldfaced text may be specified.
- Deck/card organisational metaphor - all information in WML is organised into a collection of *cards* and *decks*. Cards specify one or more units of user interaction (eg, a choice menu, a screen of text or a text entry field). Logically, a user navigates through a series of WML cards, reviews the contents of each, enters requested information, makes choices and moves on to another card.

  Cards are grouped together into decks. A WML deck is similar to an HTML page, in that it is identified by a URL [RFC1738] and is the unit of content transmission.
- Inter-card navigation and linking - WML includes support for explicitly managing the navigation between cards and decks. WML also includes provisions for event handling in the device, which may be used for navigational purposes or to execute scripts. WML also supports anchored links, similar to those found in [HTML4].
- String parameterization and state management - all WML decks can be parameterised using a state model. Variables can be used in the place of strings and are substituted at run-time. This parameterization allows for more efficient use of network resources.

2. Document Status

This document is available online in the following formats:

- PDF format at http://www.wapforum.org/.

2.1 Copyright Notice

2.2 Errata

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

2.3 Comments

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

3. References

3.1 Normative References

[ISO10646] "Information Technology - Universal Multiple-Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane", ISO/IEC 10646-1:1993.

[RFC822] "Standard for the Format of ARPA Internet Text Messages", STD 11, RFC 822, D. Crocker, August 1982. URL: ftp://ds.internic.net/rfc/rfc822.txt

[RFC1738] "Uniform Resource Locators (URL)", T. Berners-Lee, et al., December 1994. URL: ftp://ds.internic.net/rfc/rfc1738.txt

[RFC1808] "Relative Uniform Resource Locators", R. Fielding, June 1995. URL: ftp://ds.internic.net/rfc/rfc1808.txt

[RFC2045] "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", N. Freed, et al., November 1996. URL: ftp://ds.internic.net/rfc/rfc2045.txt

[RFC2048] "Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures", N. Freed, et al., November 1996. URL: ftp://ds.internic.net/rfc/rfc2048.txt

[RFC2068] "Hypertext Transfer Protocol - HTTP/1.1", R. Fielding, et al., January 1997. URL: ftp://ds.internic.net/rfc/rfc2068.txt

[RFC2119] "Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997. URL: ftp://ds.internic.net/rfc/rfc2119.txt

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

[WAE] "Wireless Application Environment Specification", WAP Forum, 30-April-1998. URL: http://www.wapforum.org/

[WAP] "Wireless Application Protocol Architecture Specification", WAP Forum, 30-April-1998. URL: http://www.wapforum.org/

[WBXML] "WAP Binary XML Content Format", WAP Forum, 30-April-1998. URL: http://www.wapforum.org/

[WSP] "Wireless Session Protocol", WAP Forum, 30-April-1998. URL: http://www.wapforum.org/

[XML] "Extensible Markup Language (XML), W3C Proposed Recommendation 10-February-1998, REC-xml-19980210", T. Bray, et al, February 10, 1998. URL: http://www.w3.org/TR/REC-xml

3.2 Informative References

[HDML2] "Handheld Device Markup Language Specification", P. King, et al., April 11, 1997. URL: http://www.uplanet.com/pub/hdml_w3c/hdml20-1.html

[HTML4] "HTML 4.0 Specification, W3C Recommendation 18-December-1997, REC-HTML40-971218", D. Raggett, et al., September 17, 1997. URL: http://www.w3.org/TR/REC-html40

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

4. Definitions and Abbreviations

4.1 Definitions

The following are terms and conventions used throughout this specification.

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].

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

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.

Content Encoding - when used as a verb, content encoding indicates the act of converting content from one format to another. Typically the resulting format requires less physical space than the original, is easier to process or store and/or is encrypted. When used as a noun, content encoding specifies a particular format or encoding standard or process.

Content Format - actual representation of content.

Deck - a collection of WML cards. A WML deck is also an XML document.

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 or 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.

JavaScript - a *de facto* standard language that can be used to add dynamic behaviour to HTML documents. JavaScript is one of the originating technologies of ECMAScript.

Man-Machine Interface - a synonym for user interface.

Origin Server - the server on which a given resource resides or is to be created. Often referred to as a web server or an HTTP server.

Resource - a network data object or service that can be identified by a URL. Resources may be available in multiple representations (eg, 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. A server may accept or reject a connection request from a client.

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

Terminal - a device providing the user with user agent capabilities, including the ability to request and receive information. Also called a mobile terminal or mobile station.

Transcode - the act of converting from one character set to another, eg, conversion from UCS-2 to UTF-8.

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.

WMLScript - a scripting language used to program the mobile device. WMLScript is an extended subset of the JavaScript™ scripting language.

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.

4.2 Abbreviations

For the purposes of this specification, the following abbreviations apply.

BNF	Backus-Naur Form
HDML	Handheld Markup Language [HDML2]
HTML	HyperText Markup Language [HTML4]
HTTP	HyperText Transfer Protocol [RFC2068]
IANA	Internet Assigned Number Authority
MMI	Man-Machine Interface
PDA	Personal Digital Assistant
RFC	Request For Comments
SGML	Standardised Generalised Markup Language [ISO8879]
UI	User Interface
URL	Uniform Resource Locator [RFC1738]
URN	Uniform Resource Name
W3C	World Wide Web Consortium
WAE	Wireless Application Environment [WAE]
WAP	Wireless Application Protocol [WAP]
WSP	Wireless Session Protocol [WSP]
XML	Extensible Markup Language [XML]

4.3 Device Types

WML is designed to meet the constraints of a wide range of small, narrowband devices. These devices are primarily characterised in four ways:

- Display size - smaller screen size and resolution. A small mobile device such as a phone may only have a few lines of textual display, each line containing 8-12 characters.
- Input devices - a limited, or special-purpose input device. A phone typically has a numeric keypad and a few additional function-specific keys. A more sophisticated device may have software-programmable buttons, but may not have a mouse or other pointing device.
- Computational resources - low power CPU and small memory size; often limited by power constraints.
- Narrowband network connectivity - low bandwidth and high latency. Devices with 300bps to 10kbps network connections and 5-10 second round-trip latency are not uncommon.

This document uses the following terms to define broad classes of device functionality:

- **Phone** - the typical display size ranges from two to ten lines. Input is usually accomplished with a combination of a numeric keypad and a few additional function keys. Computational resources and network throughput is typically limited, especially when compared with more general-purpose computer equipment.
- **PDA** - a Personal Digital Assistant is a device with a broader range of capabilities. When used in this document, it specifically refers to devices with additional display and input characteristics. A PDA display often supports resolution in the range of 160x100 pixels. A PDA may support a pointing device, handwriting recognition and a variety of other advanced features.

These terms are meant to define very broad descriptive guidelines and to clarify certain examples in the document.

5. WML and URLs

The World Wide Web is a network of information and devices. Three areas of specification ensure widespread interoperability:

- A unified naming model. Naming is implemented with Uniform Resource Locators (URLs), which provide standard way to name any network resource. See [RFC1738].
- Standard protocols to transport information (eg, HTTP).
- Standard content types (eg, HTML, WML).

WML assumes the same reference architecture as HTML and the World Wide Web. Content is named using URLs and is fetched over standard protocols that have HTTP semantics, such as [WSP]. URLs are defined in [RFC1738]. The character set used to specify URLs is also defined in [RFC1738].

In WML, URLs are used in the following situations:

- When specifying navigation, eg, hyperlinking.
- When specifying external resources, eg, an image or a script.

5.1 URL Schemes

WML browsers must implement the URL schemes specified in [WAE].

5.2 Fragment Anchors

WML has adopted the HTML *de facto* standard of naming locations within a resource. A WML fragment anchor is specified by the document URL, followed by a hash mark (#), followed by a fragment identifier. WML uses fragment anchors to identify individual WML cards within a WML deck. If no fragment is specified, a URL names an entire deck. In some contexts, the deck URL also implicitly identifies the first card in a deck.

5.3 Relative URLs

WML has adopted the use of relative URLs, as specified in [RFC1808]. [RFC1808] specifies the method used to resolve relative URLs in the context of a WML deck. The base URL of a WML deck is the URL that identifies the deck.

6. WML Character Set

WML is an XML language and inherits the XML document character set. In SGML nomenclature, a document character set is the set of all logical characters that a document type may contain (eg, the letter 'T' and a fixed integer identifying that letter). An SGML or XML document is simply a sequence of these integer tokens, which taken together form a document.

The document character set for XML and WML is the Universal Character Set of ISO/IEC-10646 ([ISO10646]). Currently, this character set is identical to Unicode 2.0 ([UNICODE]). WML will adopt future changes and enhancements to the [XML] and [ISO10646] specifications. Within this document, the terms ISO10646 and Unicode are used interchangeably and indicate the same document character set.

There is no requirement that WML decks be encoded using the full Unicode encoding (eg, UCS-4). Any character encoding ("charset") that contains a proper subset of the characters in Unicode may be used (eg, US-ASCII, ISO-8859-1, UTF-8, etc.). Documents not encoded using UTF-8 or UTF-16 must declare their encoding as specified in the XML specification.

6.1 Reference Processing Model

The WML reference-processing model is as follows. User agents must implement this processing model, or a model that is indistinguishable from it.

- The user agent must correctly map the external character encoding of the document to Unicode before processing the document in any way.
- Any processing of entities is done in the document character set.

A given implementation may choose any internal representation (or representations) that is convenient.

6.2 Character Entities

WML supports both named and numeric character entities. An important consequence of the reference processing model is that all numeric character entities are referenced with respect to the document character set (Unicode) and not to the current document encoding (charset).

This means that `&#302;` always refers to the same logical character, independent of the current character encoding.

WML supports the following character entity formats:

- Named character entities, such as `&` and `<`
- Decimal numeric character entities, such as `&#32;`
- Hexadecimal numeric character entities, such as `&#x20;`

Seven named character entities are particularly important in the processing of WML:

```
<!ENTITY quot  "&#34;">     <!-- quotation mark -->
<!ENTITY amp   "&#38;#38;"> <!-- ampersand -->
<!ENTITY apos  "&#39;">     <!-- apostrophe -->
<!ENTITY lt    "&#38;#60;"> <!-- less than -->
<!ENTITY gt    "&#62;">     <!-- greater than -->
<!ENTITY nbsp  " ">    <!-- non-breaking space -->
<!ENTITY shy   "&#173;">    <!-- soft hyphen (discretionary hyphen) -->
```

7. WML Syntax

WML inherits most of its syntactic constructs from XML. Refer to [XML] for in-depth information on syntactical issues.

7.1 Entities

WML text can contain numeric or named character entities. These entities specify specific characters in the document character set. Entities are used to specify characters in the document character set either which must be escaped in WML or which may be difficult to enter in a text editor. For example, the ampersand (&) is represented by the named entity `&`. All entities begin with an ampersand and end with a semicolon.

WML is an XML language. This implies that the ampersand and less-than characters must be escaped when they are used in textual data, ie, these characters may appear in their literal form only when used as markup delimiters, within a comment, etc. See [XML] for more details.

7.2 Elements

Elements specify all markup and structural information about a WML deck. Elements may contain a start tag, content and an end tag. Elements have one of two structures:

```
<tag> content </tag>
```

or

```
<tag/>
```

Elements containing content are identified by a start tag (`<tag>`) and an end tag (`</tag>`). An empty-element tag (`<tag/>`) identifies elements with no content.

7.3 Attributes

WML attributes specify additional information about an element. More specifically, attributes specify information about an element that is not part of the element's content. Attributes are always specified in the start tag of an element. For example,

```
<tag attr="abcd"/>
```

Attribute names are an XML `NAME` and are case sensitive.

XML requires that all attribute values be quoted using either double quotation marks (") or single quotation marks (’). Single quote marks can be included within the attribute value when the value is delimited by double quote marks and vice versa. Character entities may be included in an attribute value.

7.4 Comments

WML comments follow the XML commenting style and have the following syntax:

```
<!-- a comment -->
```

Comments are intended for use by the WML author and should not be displayed by the user agent. WML comments cannot be nested.

7.5 Variables

WML cards and decks can be parameterised using variables. To substitute a variable into a card or deck, the following syntaxes are used:

$*identifier*
$(*identifier***)**
$(*identifier*:*conversion***)**

Parentheses are required if white space does not indicate the end of a variable. Variable syntax has the highest priority in WML, ie, anywhere the variable syntax is legal, an unescaped '$' character indicates a variable substitution. Variable references are legal in any PCDATA and in any attribute value identified by the vdata entity type (see section 8.3).

A sequence of two dollar signs ($$) represents a single dollar sign character.

See section 10.3 for more information on variable syntax and semantics.

7.6 Case Sensitivity

XML is a case-sensitive language; WML inherits this characteristic. No case folding is performed when parsing a WML deck. This implies that all WML tags and attributes are case sensitive. In addition, any enumerated attribute values are case sensitive.

7.7 CDATA Section

CDATA sections are used to escape blocks of text and are legal in any PCDATA, eg, inside an element. CDATA sections begin with the string "<![CDATA[" and end with the string "]]>". For example:

```
<![CDATA[ this is <B> a test ]]>
```

Any text inside a CDATA section is treated as literal text and will not be parsed for markup. CDATA sections are useful anywhere literal text is convenient.

Refer to the [XML] specification for more information on CDATA sections.

7.8 Processing Instructions

WML makes no use of XML processing instructions beyond those explicitly defined in the XML specification.

7.9 Errors

The [XML] specification defines the concept of a **well-formed** XML document. WML decks that violate the definition of a well-formed document are in error. See section 14.2.2 for related information.

8. Core WML Data Types

8.1 Character Data

All character data in WML is defined in terms of XML data types. In summary:

- `CDATA` - text which may contain numeric or named character entities. `CDATA` is used only in attribute values.
- `PCDATA` - text which may contain numeric or named character entities. This text may contain tags (`PCDATA` is "Parsed `CDATA`"). `PCDATA` is used only in elements.
- `NMTOKEN` - a name token, containing any mixture of name characters, as defined by the XML specification.

See [XML] for more details.

8.2 Length

```
<!ENTITY % length  "CDATA">    <!-- [0-9]+ for pixels or [0-9]+"%" for
                                    percentage length -->
```

The `length` type may either be specified as an integer representing the number of pixels of the canvas (screen, paper) or as a percentage of the available horizontal or vertical space. Thus, the value "50" means fifty pixels. For widths, the value "50%" means half of the available horizontal space (between margins, within a canvas, etc.). For heights, the value "50%" means half of the available vertical space (in the current window, the current canvas, etc.).

The integer value consists of one or more decimal digits ([0-9]) followed by an optional percent character (%). The `length` type is only used in attribute values.

8.3 Vdata

```
<!ENTITY % vdata   "CDATA">    <!-- attribute value possibly containing
                                    variable references -->
```

The `vdata` type represents a string that may contain variable references (see section 10.3). This type is only used in attribute values.

8.4 Flow and Inline

```
<!ENTITY % layout  "BR">
<!ENTITY % inline   "%text; | %layout;">
<!ENTITY % flow     "%inline; | IMG | A">
```

The `flow` type represents "card-level" information. The `inline` type represents "text-level" information. In general, `flow` is used anywhere general markup can be included. The `inline` type indicates areas that only handle pure text or variable references.

8.5 URL

```
<!ENTITY % URL     "%vdata;">  <!-- URL or URN designating a hypertext
                                    node. May contain variable references -->
```

The `URL` type refers to either a relative or an absolute Uniform Resource Locator [RFC1738]. See section 5 for more information.

8.6 Boolean

```
<!ENTITY % boolean "(TRUE|FALSE)">
```

The `boolean` type refers to a logical value of true or false.

8.7 Number

```
<!ENTITY % number  "NMTOKEN">  <!-- a number, with format [0-9]+ -->
```

The `number` type represents an integer value greater than or equal to zero.

9. Events and Navigation

9.1 Navigation and Event Handling

WML includes navigation and event-handling models. The associated elements allow the author to specify the processing of user agent events. Events may be bound to *tasks* by the author; when an event occurs, the bound task is executed. A variety of tasks may be specified, such as navigation to an author-specified URL. Event bindings are declared by several elements, including DO and ONEVENT.

9.2 History

WML 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 URLs that represent the navigational path the user traversed to arrive at the current card. 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 card. See the NEWCONTEXT attribute (section 10.2) for more information.
- Push - a new URL is pushed onto the history stack as an effect of navigation to a new card.
- Pop - the current card's URL (top of the stack) is popped as a result of backward navigation.

The user agent must implement a navigation history. As each card is accessed via an explicitly specified URL, eg, a URL attribute in GO, the card URL is added to the history stack. The user agent must provide a means for the user to navigate back to the previous card in the history. Authors can depend on the existence of a user interface construct allowing the user to navigate backwards in the history. The user agent must return the user to the previous card in the history if a PREV task is executed (see section 9.3). The execution of PREV pops the current card URL from the history stack. Refer to section 12.5.2 for more information on the semantics of PREV.

9.3 The VAR Element

```
<!ELEMENT VAR EMPTY>
<!ATTLIST VAR
  NAME         %vdata;           #REQUIRED
  VALUE        %vdata;           #REQUIRED
  >
```

The VAR element specifies the variable to set in the current browser context as a side effect of executing a task. The element must be ignored if the NAME attribute doe not evaluate to a legal variable name at runtime (see section 10.3). See section 10.3.4 for more information on setting variables.

Attributes

NAME=*vdata*

The NAME attribute specifies the variable name.

VALUE=*vdata*

The VALUE attribute specifies the value to be assigned to the variable.

9.4 Tasks

```
<!ENTITY % task   "GO | PREV | NOOP | REFRESH">
```

Tasks specify processing that is performed in response to an event. Tasks are bound to events in the DO, ONEVENT and A elements.

9.4.1 The GO Element

```
<!ELEMENT GO (VAR)*>
<!ATTLIST GO
  URL             %URL;           #REQUIRED
  SENDREFERER     %boolean;       "FALSE"
  METHOD          (POST|GET)      "GET"
  ACCEPT-CHARSET  CDATA           #IMPLIED
  POSTDATA        %vdata;         #IMPLIED
  >
```

The GO element declares a GO task, indicating navigation to a URL. If the URL names a WML card or deck, it is displayed. A GO executes a "push" operation on the history stack (see section 9.2).

Refer to section 12.5.1 for more information on the semantics of GO.

Attributes

URL=*URL*

The URL attribute specifies the destination URL, eg, the URL of the card to display.

SENDREFERER=*boolean*

If this attribute is TRUE, the user agent must specify, for the server's benefit, the URL of the deck containing this task (ie, the referring deck). This allows a server to perform a form of access control on URLs, based on which decks are linking to them. The URL must be the smallest relative URL possible if it can be relative at all. For example, if SENDREFERER=TRUE, an HTTP based user agent shall indicate the URL of the current deck in the HTTP "Referer" request header [RFC2068].

METHOD=*(POST|GET)*

This attribute specifies the HTTP submission method. Currently, the values of GET and POST are accepted and cause the user agent to perform an HTTP GET or POST respectively. If METHOD is not specified, the user agent must use the GET method, unless the POSTDATA attribute is present, in which case the user agent must use the POST method.

ACCEPT-CHARSET=*cdata*

This 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 (charset) as specified in [RFC2045] and [RFC2068]. The IANA Character Set registry defines the public registry for charset values. This list is an exclusive-OR list, ie, the server must accept any one of the acceptable character encodings.

The default value for this attribute is the reserved string UNKNOWN. User agents should interpret this value as the character encoding that was used to transmit the WML deck containing this attribute.

POSTDATA=*vdata*

This attribute specifies data to be posted to the server. The data is sent to the server as an application/x-www-form-urlencoded entity. The data is formatted as a stream of octets encoded using the URL-escaping mechanism specified in [RFC1738].

Specifically, the following occurs:

1. The user agent should transcode the input data to the correct character set, as specified explicitly by ACCEPT-CHARSET or implicitly by the document encoding.
2. The data is escaped using URL-escaping. Any characters outside the legal URL character set will be converted into the sequence %XX, where XX is the octet represented as a hexadecimal number.
3. The resulting string is transmitted to the server in an application/x-www-form-urlencoded entity, with a charset parameter indicating the character encoding.

This attribute is ignored if the METHOD attribute has a value of GET.

9.4.2 The PREV Element

```
<!ELEMENT PREV (VAR)*>
```

The PREV element declares a PREV task, indicating navigation to the previous URL on the history stack. A PREV performs a "pop" operation on the history stack (see section 9.2).

Refer to section 12.5.2 for more information on the semantics of PREV.

9.4.3 The REFRESH Element

```
<!ELEMENT REFRESH (VAR)+>
```

The REFRESH element declares a REFRESH task, indicating an update of the user agent context as specified by the VAR elements. User-visible side effects of the state changes (eg, a change in the screen display) occur during the processing of the REFRESH task.

Refer to section 12.5.4 for more information on the semantics of REFRESH.

9.4.4 The NOOP Element

```
<!ELEMENT NOOP EMPTY>
```

This NOOP element specifies that nothing should be done, ie, "no operation".

Refer to section 12.5.3 for more information on the semantics of NOOP.

9.5 Card/Deck Task Shadowing

A variety of elements can be used to create an event binding for a card. These bindings may also be declared at the deck level:

- Card-level: the event-handling element may appear inside a CARD element and specify event-processing behaviour for that particular card.
- Deck-level: the event-handling element may appear inside a TEMPLATE element and specify event-processing behaviour for all cards in the deck. A deck-level event-handling element is equivalent to specifying the event-handling element in each card.

A card-level event-handling element overrides (or "shadows") a deck-level event-handling element if they both specify the same event. A card-level ONEVENT element will shadow a deck-level ONEVENT element if they both have the same TYPE. A card-level DO element will shadow a deck-level DO element if they have the same NAME.

If a card-level element shadows a deck-level element and the card-level element specifies the NOOP task, the event binding for that event will be completely masked. In this situation, the card- and deck-level element will be ignored and no side effects will occur on delivery of the event. In this case, user agents should not expose the element to the user (eg, render a UI control). In effect, the NOOP removes the element from the card.

In the following example, a deck-level DO element indicates that a PREV task should execute on receipt of a particular user action. The first card inherits the DO element specified in the TEMPLATE element and will display the DO to the user. The second card shadows the deck-level DO element with a NOOP. The user agent will not display the DO element when displaying the second card. The third card shadows the deck-level DO element, causing the user agent to display the alternative label and to perform the GO task if the DO is selected.

```
<WML>
  <TEMPLATE>
    <DO TYPE="OPTIONS" NAME="do1" LABEL="default">
      <PREV/>
    </DO>
  </TEMPLATE>

  <CARD NAME="first">
    <!-- deck-level DO not shadowed.  The card exposes the
         deck-level DO as part of the current card -->

    <!-- rest of card -->
    …
  </CARD>

  <CARD NAME="second">
    <!-- deck-level DO is shadowed with NOOP.
         It is not exposed to the user -->
    <DO TYPE="OPTIONS" NAME="do1">
      <NOOP/>
    </DO>

    <!-- rest of card -->
    …
  </CARD>

  <CARD NAME="third">
    <!-- deck-level DO is shadowed. It is replaced by a card-level DO -->
    <DO TYPE="OPTIONS" NAME="do1" LABEL="options">
      <GO URL="/options"/>
    </DO>

    <!-- rest of card -->
    …
  </CARD>
</WML>
```

9.6 The DO Element

```
<!ENTITY % task   "GO | PREV | NOOP | REFRESH">
<!ELEMENT DO (%task;)>
<!ATTLIST DO
  TYPE         CDATA       #REQUIRED
  LABEL        %vdata;     #IMPLIED
  NAME         NMTOKEN     #IMPLIED
  OPTIONAL     %boolean;   "FALSE"
  >
```

The DO element provides a general mechanism for the user to act upon the current card, ie, a card-level user interface element. The representation of the DO element is user agent dependent and the author must only assume that the element is mapped to a unique user interface *widget* that the user can activate. For example, the widget mapping may be to a graphically rendered button, a soft or function key, a voice-activated command sequence, or any other interface that has a simple "activate" operation with no inter-operation persistent state.

The TYPE attribute is provided as a hint to the user agent about the author's intended use of the element and should be used by the user agent to provide a suitable mapping onto a physical user interface construct. WML authors must not rely on the semantics or behaviour of an individual TYPE value, or on the mapping of TYPE to a particular physical construct.

The DO element may appear at both the card and deck-level:

- Card-level: the DO element may appear inside a CARD element and may be located anywhere in the text flow. If the user agent intends to render the DO element inline (ie, in the text flow), it should use the element's anchor point as the rendering point. WML authors must not rely on the inline rendering of the DO element and must not rely on the correct positioning of an inline rendering of the element.
- Deck-level: the DO element may appear inside a TEMPLATE element, indicating a deck-level DO element. A deck-level DO element applies to all cards in the deck (ie, is equivalent to having specified the DO within each card). For the purposes of inline rendering, the user agent must behave as if deck-level DO elements are located at the end of the card's text flow.

A card-level DO element overrides (or "shadows") a deck-level DO element if they have the same NAME (see section 9.5 for more details). For a single card, the *active* DO elements are defined as the DO elements specified in the card, plus any DO elements specified in the deck's TEMPLATE and not overridden in the card. All active DO elements with a NOOP task must not be presented to the user. All DO elements that are not active must not be presented to the user. All DO elements with a task other than NOOP must be made accessible to the user in some manner. In other words, it must be possible for the user to activate these user interface items when viewing the card containing the active DO elements. When the user activates the DO element, the associated task is executed.

Attributes

TYPE=*cdata*

The DO element type. This attribute provides a hint to the user agent about the author's intended use of the element and how the element should be mapped to a physical user interface construct. All types are reserved, except for those marked as experimental.

User agents must accept any TYPE, but may treat any unrecognised type as the equivalent of UNKNOWN.

In the following table, the * character represents any string, eg, Test* indicates any string starting with the word Test.

Table 1. Pre-defined DO types

Type	***Description***
ACCEPT	Positive acknowledgement (acceptance)
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. Equivalent to an empty string (eg, TYPE="").
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 DO element type. See [RFC2045] for more information.

LABEL=*vdata*

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

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

NAME=*nmtoken*

This attribute specifies the name of the DO event binding. If two DO elements are specified with the same name, they refer to the same binding. If DO elements are specified both at the card-level (in a CARD element) and at the deck-level (in a TEMPLATE element) and both elements have the same NAME, the deck-level DO element is ignored. It is an error to specify two or more DO elements with the same NAME in a single card or in the TEMPLATE element. A NAME with an empty value is equivalent to an unspecified NAME attribute. An unspecified NAME defaults to the value of the TYPE attribute.

OPTIONAL=*boolean*

If this attribute has a value of TRUE, the user agent may ignore this element.

9.7 The A Element

```
<!ELEMENT A ( %inline; | GO | PREV | REFRESH )*>
<!ATTLIST A
  TITLE        %vdata;      #IMPLIED
  >
```

The A element specifies the head of a link. The tail of a link is specified as part of other elements (eg, a card name attribute). It is an error to nest anchored links.

Anchors may be present in any text flow, excluding the text in OPTION elements (ie, anywhere formatted text is legal, except for OPTION elements). Anchored links have an associated *task* that specifies the behaviour when the anchor is selected. It is an error to specify more than one task element (eg, GO, PREV or REFRESH) in an A element.

Attributes

TITLE=*vdata*

This attribute specifies a brief text string identifying the link. The user agent may display it in a variety of ways, including dynamic labelling of a button or key, a *tool tip*, a voice prompt, etc. The user agent may truncate or ignore this attribute depending on the characteristics of the navigational user interface. To work well on a broad range of user agents, the author should limit all labels to 6 characters in length.

9.8 Intrinsic Events

Several WML elements are capable of generating events when the user interacts with them. These so-called "intrinsic events" indicate state transitions inside the user agent. Individual elements specify the events they can generate. WML defines the following intrinsic events:

Table 2. WML Intrinsic Events

Event	***Element(s)***	***Description***
ONTIMER	CARD, TEMPLATE	The ONTIMER event occurs when a timer expires. Timers are specified using the TIMER element (see section 11.7).
ONENTERFORWARD	CARD, TEMPLATE	The ONENTERFORWARD event occurs when the user causes the user agent to enter a card using a GO task or any method with identical semantics. This includes card entry caused by a script function or user-agent-specific mechanisms, such as a means to directly enter and navigate to a URL.

Event	***Element(s)***	***Description***
		The ONENTERFORWARD intrinsic event may be specified at both the card and deck-level. Event bindings specified in the TEMPLATE element apply to all cards in the deck and may be overridden as specified in section 9.5.
ONENTERBACKWARD	CARD, TEMPLATE	The ONENTERBACKWARD event occurs when the user causes the user agent to navigate into a card using a PREV task or any method with identical semantics. In other words, the ONENTERBACKWARD event occurs when the user causes the user agent to navigate into a card by using a URL retrieved from the history stack. This includes navigation caused by a script function or user-agent-specific mechanisms. The ONENTERBACKWARD intrinsic event may be specified at both the card and deck-level. Event bindings specified in the TEMPLATE element apply to all cards in the deck and may be overridden as specified in section 9.5.
ONCLICK	OPTION	The ONCLICK event occurs when the user selects or deselects this item.

The author may specify that certain tasks are to be executed when an intrinsic event occurs. This specification may take one of two forms. The first form specifies a URL to be navigated to when the event occurs. This event binding is specified in a well-defined element-specific attribute and is the equivalent of a GO task. For example:

```
<CARD ONENTERFORWARD="/url"> Hello </CARD>
```

This attribute value may only specify a URL.

The second form is an expanded version of the previous, allowing the author more control over user agent behaviour. An ONEVENT element is declared within a parent element, specifying the full event binding for a particular intrinsic event. For example, the following is identical to the previous example:

```
<CARD>
  <ONEVENT TYPE="ONENTERFORWARD">
    <GO URL="/url"/>
  </ONEVENT>
  Hello
</CARD>
```

The user agent must treat the attribute syntax as an abbreviated form of the ONEVENT element where the attribute name is mapped to the ONEVENT type.

An intrinsic event binding is scoped to the element in which it is declared, eg, an event binding declared in a card is local to that card. Any event binding declared in an element is active only within that element. Event bindings specified in sub-elements take precedence over any conflicting event bindings declared in a parent element. Conflicting event bindings within an element are an error.

9.8.1 The ONEVENT Element

```
<!ENTITY % task   "GO | PREV | NOOP | REFRESH">
<!ELEMENT ONEVENT (%task;)>
<!ATTLIST ONEVENT
  TYPE         CDATA       #REQUIRED
  >
```

The ONEVENT element binds a task to a particular intrinsic event for the immediately enclosing element, ie, specifying an ONEVENT element inside an "XYZ" element associates an intrinsic event binding with the "XYZ" element.

The user agent must ignore any ONEVENT element specifying a TYPE that does not correspond to a legal intrinsic event for the immediately enclosing element.

Attributes

TYPE=*cdata*

The TYPE attribute indicates the name of the intrinsic event.

9.8.2 Card/Deck Intrinsic Events

The ONENTERFORWARD and ONENTERBACKWARD intrinsic events may be specified at both the card- and deck-level and have the shadowing semantics defined in section 9.5. Intrinsic events may be overridden regardless of the syntax used to specify them. A deck-level event-handler specified with the ONEVENT element may be overridden by the ONEVENTFORWARD attribute and vice versa.

10. The State Model

WML includes support for managing user agent state, including:

- Variables - parameters used to change the characteristics and content of a WML card or deck;
- History - navigational history, which may be used to facilitate efficient backward navigation; and
- Implementation-dependent state - other state relating to the particulars of the user agent implementation and behaviour.

10.1 The Browser Context

WML state is stored in a single scope, known as a *browser context.* The browser context is used to manage all parameters and user agent state, including variables, the navigation history and other implementation-dependent information related to the current state of the user agent.

10.2 The NEWCONTEXT Attribute

The browser context may be initialised to a well-defined state by the `NEWCONTEXT` attribute of the `CARD` element (see section 11.5). This attribute indicates that the browser context should be re-initialised and must perform the following operations:

- Unset (remove) all variables defined in the current browser context,
- Clear the navigational history state, and
- Reset implementation-specific state to a well-known value.

`NEWCONTEXT` is only performed as part of the `GO` task. See section 12.5 for more information on the processing of state during navigation.

10.3 Variables

All WML content can be parameterised, allowing the author a great deal of flexibility in creating cards and decks with improved caching behaviour and better perceived interactivity. 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 value is *not set* if it has a value equal to the empty string, or is otherwise unknown or undefined in the current browser context.

10.3.1 Variable Substitution

The values of variables can be substituted into both the text (`#PCDATA`) of a card and into `%vdata` and `%URL` attribute values in WML elements. Only textual information can be substituted; no substitution of elements or attributes is possible. The substitution of variable values happens at run-time in the user agent. 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.

WML variable names consist of an 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 is a BNF-like description of the variable substitution syntax. The description uses the conventions established in [RFC822], except that the "`|`" character is used to designate alternatives. Briefly, "`(`" and "`)`" are used to group elements, optional elements are enclosed in "`[`" and "`]`". Elements may be preceded with `<N>*` to specify N or more repetitions of the following element (`N` defaults to zero when unspecified).

```
var     = ( "$" varname ) |
          ( "$(" varname [ conv ] ")" )

conv    = ":" ( escape | noesc | unesc )
escape  = ("E" | "e") [ ( "S" | "s" ) ( "C" | "c" )
                        ( "A" | "a" ) ( "P" | "p" )
                        ( "E" | "e" ) ]
noesc   = ( "N" | "n" ) [ ( "O" | "o" ) ("E" | "e")
                          ( "S" | "s" ) ( "C" | "c" ) ]
unesc   = ( "U" | "u" ) [ ( "N" | "n" ) ("E" | "e")
                          ( "S" | "s" ) ( "C" | "c" ) ]

varname = ( "_" | alpha ) *[ "_" | alpha | digit ]
alpha   = lalpha | halpha
lalpha  = "a" | "b" | "c" | "d" | "e" | "f" | "g" | "h" | "i" |
          "j" | "k" | "l" | "m" | "n" | "o" | "p" | "q" | "r" |
          "s" | "t" | "u" | "v" | "w" | "x" | "y" | "z"
halpha  = "A" | "B" | "C" | "D" | "E" | "F" | "G" | "H" | "I" |
          "J" | "K" | "L" | "M" | "N" | "O" | "P" | "Q" | "R" |
          "S" | "T" | "U" | "V" | "W" | "X" | "Y" | "Z"
digit   = "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" |
          "8" | "9"
```

Parentheses are required anywhere the end of a variable cannot be inferred from the surrounding context, eg, an illegal character such as white space.

For example:

```
This is a $var
This is another $(var).
This is an escaped $(var:e).
Long form of escaped $(var:escape).
Long form of unescape $(var:unesc).
Short form of no-escape $(var:N).
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 summarised the current conversions and their legal abbreviations:

Table 3. Variable escaping methods

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

The use of a conversion during variable substitution does not affect the actual value of the variable.

URL-escaping is detailed in [RFC1738]. All lexically sensitive characters defined in WML must be escaped, including all `reserved` and `unsafe` URL characters, as specified by [RFC1738].

If no conversion is specified, the variable is substituted using the conversion format appropriate for the context. The `ONENTERBACKWARD`, `ONENTERFORWARD`, `URL` and `SRC` attributes default to escape conversion, elsewhere no conversion is done. Specifying the `noesc` conversion disables context sensitive escaping of a variable.

10.3.2 Parsing the Variable Substitution Syntax

The variable substitution syntax (eg, $X) is parsed after all XML parsing is complete. In XML terminology, variable substitution is parsed 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 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 syntax, eg, entity substitution occurs before the variable substitution syntax is parsed. The following examples are identical references to the variable named X:

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

10.3.3 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. A single dollar-sign entity, even specified as $, results in a variable substitution.

In order to include a '$' character in a WML deck, 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 displayed as:

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

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

```
%24
```

10.3.4 Setting Variables

There are a number of ways to set the value of a variable. When a variable is set and it is already defined in the browser context, the current value is updated.

The VAR element allows the author to set variable state as a side effect of navigation. VAR may be specified in task elements, including GO, PREV and REFRESH. The VAR element specifies a variable name and value, for example:

```
<VAR NAME="location" VALUE="$(X)"/>
```

The variable specified in the NAME attribute (eg, location) is set as a side effect of navigation. See the discussion of event handling (section 9 and section 12.5) for more information on the processing of the VAR element.

Input elements set the variable identified by the KEY attribute to any information entered by the user. For example, an INPUT element assigns the entered text to the variable, and the SELECT element assigns the value present in the VALUE attribute of the chosen OPTION element.

User input is written to variables when the user commits the input to the INPUT or SELECT element. Committing input is an MMI dependent concept, and the WML author must not rely on a particular user interface. For example, some implementations will update the variable with each character entered into an INPUT element, and others will defer the variable update until the INPUT element has lost focus. The user agent must update all variables prior to the execution of any task. The user agent may re-display the current card when variables are set, but the author must not assume that this action will occur.

11. The Structure of WML Decks

WML data are structured as a collection of *cards*. A single collection of cards is referred to as a WML *deck*. Each card contains structured content and navigation specifications. Logically, a user navigates through a series of cards, reviews the contents of each, enters requested information, makes choices and navigates to another card or returns to a previously visited card.

11.1 Document Prologue

A valid WML deck is a valid XML document and therefore must contain an XML declaration and a document type declaration (see [XML] for more detail about the definition of a valid document). A typical document prologue contains:

```
<?xml version="1.0"?>
<!DOCTYPE WML PUBLIC "-//WAPFORUM//DTD WML 1.0//EN"
                     "http://www.wapforum.org/DTD/wml.xml">
```

It is an error to omit the prologue.

11.2 The WML Element

```
<!ELEMENT WML ( HEAD?, TEMPLATE?, CARD+ )>
<!ATTLIST WML
  xml:lang        NMTOKEN        #IMPLIED
  >
```

The WML element defines a deck and encloses all information and cards in the deck.

Attributes

`xml:lang=`*`nmtoken`*

The `xml:lang` attribute specifies the natural or formal language in which the document is written. See [XML] for details on the syntax and specification of the attribute values. If the `xml:lang` attribute is specified, it takes precedence over any other specification of the document language (eg, transport meta data).

11.2.1 A WML Example

The following is a deck containing two cards, each represented by a CARD element (see section 11.5 for information on cards). After loading the deck, a user agent displays the first card. If the user activates the DO element, the user agent displays the second card.

```
<WML>
  <CARD>
    <DO TYPE="ACCEPT">
      <GO URL="#card2"/>
    </DO>
    Hello world!
    This is the first card...
  </CARD>

  <CARD NAME="card2">
    This is the second card.
    Goodbye.
  </CARD>
</WML>
```

11.3 The HEAD Element

```
<!ELEMENT HEAD ( ACCESS | META )+>
```

The HEAD element contains information relating to the deck as a whole, including meta-data and access control elements.

11.3.1 The ACCESS Element

```
<!ELEMENT ACCESS EMPTY>
<!ATTLIST ACCESS
  DOMAIN       CDATA       #IMPLIED
  PATH         CDATA       #IMPLIED
  PUBLIC       %boolean;   "FALSE"
  >
```

The ACCESS element specifies access control information for the entire deck. It is an error for a deck to contain more than one ACCESS element.

Attributes

DOMAIN=*cdata*
PATH=*cdata*

A deck's DOMAIN and PATH attributes specify which other decks may access it. As the user agent navigates from one deck to another, it performs access control checks to determine whether the destination deck allows access from the current deck.

If a deck has a DOMAIN and/or PATH attribute, the referring deck's URL must match the values of the attributes. Matching is done as follows: the access domain is suffix-matched against the domain name portion of the referring URL and the access path is prefix matched against the path portion of the referring URL.

DOMAIN suffix matching is done using the entire element of each sub-domain and must match each element exactly (eg, `www.wapforum.org` shall match `wapforum.org`, but shall not match `forum.org`). PATH prefix matching is done using entire path elements and must match each element exactly (eg, `/X/Y` matches `/X`, but does not match `/XZ`).

The DOMAIN attribute defaults to the current deck'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 deck, the PATH attribute accepts relative URLs. 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 URLs would be allowed to go to the deck:

```
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 URLs would not be allowed to go to the deck:

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

DOMAIN and PATH follow URL capitalisation rules.

PUBLIC=*boolean*

This attribute indicates whether deck access control has been disabled for this deck. If disabled, ie, PUBLIC="TRUE" is specified, cards in any deck can access this deck. If enabled, then the DOMAIN and PATH attributes are used to determine which cards or decks can access the deck. By default, access control is enabled.

11.3.2 The META Element

```
<!ELEMENT META EMPTY>
<!ATTLIST META
  HTTP-EQUIV     CDATA      #IMPLIED
  NAME           CDATA      #IMPLIED
  USER-AGENT     CDATA      #IMPLIED
  CONTENT        CDATA      #REQUIRED
  SCHEME         CDATA      #IMPLIED
  >
```

The META element contains generic meta-information relating to the WML deck. Meta-information is specified with property names and values. This specification does not define any properties, nor does it define how user agents must interpret meta-data. User agents are not required to support the meta-data mechanism.

It is an error for a META element to contain more than one attribute specifying a property name, ie, more than one attribute from the following set: NAME, HTTP-EQUIV and USER-AGENT.

Attributes

NAME=*cdata*

This attribute specifies the property name. The user agent must ignore any meta-data named with this attribute. Network servers should not emit WML content containing meta-data named with this attribute.

HTTP-EQUIV=*cdata*

This attribute may be used in place of NAME and indicates that the property should be interpreted as an HTTP header (see [RFC2068]). Meta-data named with this attribute should be converted to a WSP or HTTP response header if the content is tokenized before it arrives at the user agent.

USER-AGENT=*cdata*

This attribute may be used in place of NAME. This meta-data must be delivered to the user agent and may not be removed by any network intermediary.

CONTENT=*cdata*

This attribute specifies the property value.

SCHEME=*cdata*

This attribute specifies a form or structure that may be used to interpret the property value. Scheme values vary depending on the type of meta-data.

11.4 The TEMPLATE Element

```
<!ENTITY % navelmts "DO | ONEVENT">
<!ELEMENT TEMPLATE (%navelmts;)*>
<!ATTLIST TEMPLATE
  %cardev;
  >
```

The TEMPLATE element declares a template for cards in the deck. Event bindings specified in the TEMPLATE element (eg, DO or ONEVENT) apply to all cards in the deck. Specifying an event binding in the TEMPLATE element is equivalent to specifying it in every card element. A card element may override the behaviour specified in the TEMPLATE element. In particular:

- DO elements specified in the TEMPLATE element may be overridden in individual cards if both elements have the same NAME attribute value. See section 9.5 for more information.
- Intrinsic event bindings specified in the TEMPLATE element may be overridden by the specification of an event binding in a card element. See section 9.8 for more information.

See section 11.5 for the definition of the card-level intrinsic events (the cardev entity).

Attributes Defined Elsewhere

The following task attributes are defined in section 11.5.1:

%cardev

11.5 The Card Element

A WML deck contains a collection of cards. There is a variety of card types, each specifying a different mode of user interaction.

11.5.1 Card Intrinsic Events

```
<!ENTITY % cardev
 "ONENTERFORWARD  %URL;          #IMPLIED
  ONENTERBACKWARD %URL;          #IMPLIED
  ONTIMER         %URL;          #IMPLIED"
  >
```

The following attributes are available in the CARD and TEMPLATE elements.

Attributes

ONENTERFORWARD=*URL*

The ONENTERFORWARD event occurs when the user causes the user agent to navigate into a card using a GO task.

ONENTERBACKWARD=*URL*

The ONENTERBACKWARD event occurs when the user causes the user agent to navigate into a card using a PREV task.

ONTIMER=*URL*

The ONTIMER event occurs when a TIMER expires.

11.5.2 The CARD Element

```
<!ENTITY % fields  "%flow; | INPUT | SELECT | FIELDSET">
<!ELEMENT CARD (%fields; | %navelmts; | TIMER)*>
<!ATTLIST CARD
  NAME            NMTOKEN       #IMPLIED
  TITLE           %vdata;       #IMPLIED
  NEWCONTEXT      %boolean;     "FALSE"
  STYLE           (LIST|SET)    "LIST"
  %cardev;
  >
```

The CARD element is a container of text and input elements that is sufficiently flexible to allow presentation and layout in a wide variety of devices, with a wide variety of display and input characteristics. The CARD element indicates the general layout and required input fields, but does not overly constrain the user agent implementation in the areas of layout or user input. For example, a CARD can be presented as a single page on a large-screen device and as a series of smaller pages on a small-screen device.

A CARD can contain markup, input fields and elements indicating the structure of the card. The order of elements in the card is significant and should be respected by the user agent.

Attributes

NAME=*nmtoken*

This attribute gives a name to the card. A card's name may be used as a fragment anchor. See section 5.2 for more information.

TITLE=*vdata*

The TITLE attribute specifies advisory information about the card. The title may be rendered in a variety of ways by the user agent (eg, suggested bookmark name, pop-up *tooltip*, etc.).

NEWCONTEXT=*boolean*

This attribute indicates that the current browser context should be re-initialised upon entry to this card. See section 10.2 for more information.

STYLE=*(LIST|SET)*

This attribute specifies a hint to the user agent about the organisation of the CARD content. This hint may be used to organise the content presentation or to otherwise influence layout of the card.

- LIST - the card is naturally organised as a linear sequence of field elements, eg, a set of questions or fields which are naturally handled by the user in the order in which they are specified in the group. This style is best for short forms in which no fields are optional (eg, sending an email message requires a To: address, a subject and a message, and they are logically specified in this order).

  It is expected that in small-screen devices, LIST groups may be presented as a sequence of screens, with a screen flip in between each field or fieldset. Other user agents may elect to present all fields simultaneously.

- SET - the card is a collection of field elements without a natural order. This is useful for collections of fields containing optional or unordered components or simple record data where the user is updating individual input fields.

  It is expected that in small-screen devices, SET groups may be presented by using a hierarchical or tree organisation. In these types of presentation, the TITLE attribute of each field and fieldset may be used to define the name presented to the user in the top-level summary card.

The user agent may interpret the style attribute in a manner appropriate to its device capabilities (eg, screen size or input device). In addition, the user agent should adopt user interface conventions for handling the editing of input elements in a manner that best suits the device's input model.

For example, a phone-class device displaying a CARD with STYLE=SET may use a softkey or button to select individual fields for editing or viewing. A PDA-class device might create soft buttons on demand, or simply present all fields on the screen for direct manipulation.

On devices with limited display capabilities, it is often necessary to insert screen flips or other user-interface transitions between fields. When this is done, the user agent needs to decide on the proper boundary between fields. User agents may use the following heuristic for determining the choice of a screen flip location:

- FIELDSET defines a logical boundary between fields.
- Fields (eg, INPUT) may be individually displayed. When this is done, the line of markup (flow) immediately preceding the field should be treated as a field prompt and displayed with the input element.

Attributes Defined Elsewhere

The following task attributes are defined in section 11.5.1:

%cardev

11.5.2.1 A CARD Example

The following is an example of a simple CARD element embedded within a WML deck. The card contains text, which is displayed by the user agent. In addition, the example demonstrates the use of a simple DO element, defined at the deck level.

```
<WML>
   <TEMPLATE>
      <DO TYPE="ACCEPT" LABEL="Exit">
         <PREV/>
      </DO>
   </TEMPLATE>
   <CARD>
      Hello World!
   </CARD>
</WML>
```

11.6 Control Elements

11.6.1 The TABINDEX Attribute

Attributes

TABINDEX=*number*

This attribute specifies the tabbing position of the current element. The tabbing position indicates the relative order in which elements are traversed when tabbing within a single WML card. A numerically greater TABINDEX value indicates an element that is later in the tab sequence than an element with a numerically lesser TABINDEX value.

Each input element (ie, INPUT and SELECT) in a card is assigned a position in the card's tab sequence. In addition, the user agent may assign a tab position to other elements. The TABINDEX attribute indicates the tab position of a given element. Elements that are not designated with an author-specified tab position may be assigned one by the user agent. User agent specified tab positions must be later in the tab sequence than any author-specified tab positions.

Tabbing is a navigational accelerator and is optional for all user agents. Authors must not assume that a user agent implements tabbing.

11.6.2 Select Lists

Select lists are an input element that specifies a list of options for the user to choose from. Single and multiple choice lists are supported.

11.6.2.1 The SELECT Element

```
<!ELEMENT SELECT (OPTGROUP|OPTION)+>
<!ATTLIST SELECT
  TITLE        %vdata;          #IMPLIED
  KEY          NMTOKEN          #IMPLIED
  DEFAULT      %vdata;          #IMPLIED
  IKEY         NMTOKEN          #IMPLIED
  IDEFAULT     %vdata;          #IMPLIED
  MULTIPLE     %boolean;        "FALSE"
  TABINDEX     %number;         #IMPLIED
  >
```

The SELECT element lets users pick from a list of options. Each option is specified by an OPTION element. Each OPTION element may have one line of formatted text (which may be wrapped or truncated by the user agent if too long). OPTION elements may be organised into hierarchical groups using the OPTGROUP element.

Attributes

MULTIPLE=*boolean*

This attribute indicates that the select list should accept multiple selections. When not set, the select list should only accept a single selected option.

KEY=*nmtoken*
DEFAULT=v*data*

This KEY attribute indicates the name of the variable to set with the result of the selection. The variable is set to the string value of the chosen OPTION element, which is specified with the VALUE attribute. The KEY variable's value is used to pre-select options in the select list.

The DEFAULT attribute indicates the default value of the variable named in the KEY attribute. When the element is displayed, and the variable named in the KEY attribute is not set, the KEY variable is assigned the value specified in the DEFAULT attribute. If the KEY variable already contains a value, the DEFAULT attribute is ignored. Any application of the default value is done before the list is pre-selected with the value of the KEY variable.

If this element allows the selection of multiple options, the result of the user's choice is a list of all selected values, separated by the semicolon character. The KEY variable is set with this result. In addition, the DEFAULT attribute is interpreted as a semicolon-separated list of pre-selected options.

IKEY=*nmtoken*
IDEFAULT=*vdata*

The IKEY attribute indicates the name of the variable to be set with the index result of the selection. The index result is the position of the currently selected OPTION in the select list. An index of zero indicates that no OPTION is selected. Index numbering begins at one and increases monotonically.

The IDEFAULT attribute indicates the default-selected OPTION element. When the element is displayed, if the variable named in the IKEY attribute is not set, it is assigned the default-selected entry. If the variable already contains a value, the IDEFAULT attribute is ignored. If the IKEY attribute is not specified, the IDEFAULT value is applied every time the element is displayed.

If this element allows the selection of multiple options, the index result of the user's choice is a list of the indices of all the selected options, separated by the semicolon character (eg, "1;2"). The IKEY variable is set with this result. In addition, the IDEFAULT attribute is interpreted as a semicolon-separated list of pre-selected options (eg, "1;4").

TITLE=*vdata*

This attribute specifies a title for this element, which may be used in the presentation of this object.

Attributes Defined Elsewhere

The following attribute is defined in section 11.6.1:

TABINDEX

On entry into a card containing a SELECT element, the user agent must select the initial options in the following way:

- If the IKEY attribute exists, the indices in the variable named by IKEY are used to select the option. If the specified variable is not set, the index is assumed to be 1. If any index is larger than the number of options in the select list, the last entry is selected.
- If the IKEY attribute does not exist and the KEY attribute exists, the value of the variable specified by KEY is used to select options. If the variable specified by KEY is not set or no OPTION has a VALUE attribute matching the value, the first option is selected.

Once an OPTION is selected, the variable named by KEY is updated to the value of the option.

Both KEY and IKEY, or DEFAULT and IDEFAULT may be specified. IDEFAULT takes precedence over DEFAULT and IKEY takes precedence over KEY.

11.6.2.2 The OPTION Element

```
<!ELEMENT OPTION (%text; | ONEVENT)*>
<!ATTLIST OPTION
  VALUE      %vdata;    #IMPLIED
  TITLE      %vdata;    #IMPLIED
  ONCLICK    %URL;      #IMPLIED
  >
```

This element specifies a single choice option in a SELECT element.

Attributes

VALUE=*vdata*

The VALUE attribute specifies the value to be used when setting the KEY variable. When the user selects this option, the resulting value specified in the VALUE attribute is used to set the SELECT element's KEY variable.

The VALUE attribute may contain variable references, which are evaluated before the KEY variable is set.

TITLE=*vdata*

This attribute specifies a title for this element, which may be used in the presentation of this object.

ONCLICK=*URL*

The ONCLICK event occurs when the user selects or deselects this option. A multiple-selection option list generates an ONCLICK event whenever the user selects or deselects this option. A single-selection option list generates an ONCLICK event when the user selects this option, ie, no event is generated for the de-selection of any previously selected option.

11.6.2.3 The OPTGROUP Element

```
<!ELEMENT OPTGROUP (OPTGROUP|OPTION)+ >
<!ATTLIST OPTGROUP
  TITLE      %vdata;    #IMPLIED
  >
```

The OPTGROUP element allows the author to group related OPTION elements into a hierarchy. The user agent may use this hierarchy to facilitate layout and presentation on a wide variety of devices.

Attributes

TITLE=*vdata*

This attribute specifies a title for this element, which may be used in the presentation of this object.

11.6.2.4 Select list examples

In this example, a simple single-choice select list is specified. If the user were to choose the "Dog" option, the variable "X" would be set to a value of "D".

```
<WML>
  <CARD>
    Please choose your favourite animal:
    <SELECT KEY="X">
      <OPTION VALUE="D">Dog</OPTION>
      <OPTION VALUE="C">Cat</OPTION>
    </SELECT>
  </CARD>
</WML>
```

In this example, a single choice select list is specified. If the user were to choose the "Cat" option, the variable "I" would be set to a value of "2". In addition, the "Dog" option would be pre-selected if the "I" variable had not been previously set.

```
<WML>
  <CARD>
    Please choose your favourite animal:
    <SELECT IKEY="I" IDEFAULT="1">
      <OPTION VALUE="D">Dog</OPTION>
      <OPTION VALUE="C">Cat</OPTION>
    </SELECT>
  </CARD>
</WML>
```

In this example, a multiple-choice list is specified. If the user were to choose the "Cat" and "Horse" options, the variable "X" would be set to "C;H" and the variable "I" would be set to "1;3". In addition, the "Dog" and "Cat" options would be pre-selected if the variable "I" had not been previously set.

```
<WML>
  <CARD>
    Please choose <I>all</I> of your favourite animals:
    <SELECT KEY="X" IKEY="I" IDEFAULT="1;2" MULTIPLE="TRUE">
      <OPTION VALUE="D">Dog</OPTION>
      <OPTION VALUE="C">Cat</OPTION>
      <OPTION VALUE="H">Horse</OPTION>
    </SELECT>
  </CARD>
</WML>
```

11.6.3 The INPUT Element

```
<!ELEMENT INPUT EMPTY>
<!ATTLIST INPUT
  KEY        NMTOKEN           #REQUIRED
  TYPE       (TEXT|PASSWORD)   "TEXT"
  VALUE      %vdata;           #IMPLIED
  DEFAULT    %vdata;           #IMPLIED
  FORMAT     CDATA             #IMPLIED
  EMPTYOK    %boolean;         "FALSE"
  SIZE       %number;          #IMPLIED
  MAXLENGTH  %number;          #IMPLIED
  TABINDEX   %number;          #IMPLIED
  TITLE      %vdata;           #IMPLIED
  >
```

The INPUT element specifies a text entry object. The user input is constrained by the optional FORMAT attribute.

Attributes

KEY=*nmtoken*
DEFAULT=v*data*
VALUE=*vdata*

The KEY attribute specifies the name of the variable to set with the result of the user's text input. The KEY variable's value is used to pre-load the text entry object.

The DEFAULT attribute indicates the default value of the variable named in the KEY attribute. When the element is displayed and the variable named in the KEY attribute is not set, the KEY variable is assigned the value specified in the DEFAULT attribute. If the KEY variable already contains a value, the DEFAULT attribute is ignored. If the DEFAULT attribute specifies a value that does not conform to the input mask specified by the FORMAT attribute, the user agent must ignore the DEFAULT attribute.

The DEFAULT and VALUE attributes are identical in their behaviour and syntax.

TYPE=*(TEXT|PASSWORD)*

This attribute specifies the type of text-input area. The default type is TEXT. The following values are allowed:

- TEXT - a text entry box. Input should be displayed to the user in a readable form and each character should be echoed in a manner appropriate to the user agent.
- PASSWORD - a text entry box. Input of each character should be echoed in an obscured or illegible form. For example, user agents may elect to display an asterisk in place of a character entered by the user. Typically, the PASSWORD input mode is indicated for password entry or other private data. Note that PASSWORD input is not secure and should not be depended on for critical applications.

In both cases, the user's input is applied to the KEY variable.

FORMAT=*cdata*

The FORMAT attribute specifies an input mask for user input entries. The string consists of mask control characters and static text that is displayed in the input area. The user agent may use the format mask to facilitate accelerated data input.

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 upper-case alphabetic or punctuation character (ie, upper-case non-numeric character)

a entry of any lower-case alphabetic or punctuation character (ie, lower-case non-numeric character)

N entry of any numeric character

X entry of any upper case character

x entry of any lower-case character

M entry of any character; the user agent may choose to assume that the character is upper-case for the purposes of simple data entry, but must allow entry of any character

m entry of any character; the user agent may choose to assume that the character is lower-case for the purposes of simple data entry, but must allow entry of any character

***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 n characters where n is from 1 to 9; 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*

c display the next character, c, in the entry field; allows quoting of the format codes so they can be displayed in the entry area

User agents must implement the format codes to the best of their ability given the constraints of the input language and character set. If the input language and character set have a clear definition of numbers and character case, they must be followed. Authors must not rely on the interpretation of a particular format code in a given language.

EMPTYOK=*boolean*

The EMPTYOK attribute indicates that this INPUT element accepts empty input although a non-empty format string has been specified. Typically, the EMPTYOK attribute is indicated for formatted entry fields that are optional. By default, INPUT elements specifying a FORMAT require the user to input data matching the FORMAT specification.

SIZE=*number*

This attribute specifies the width, in characters, of the text-input area. The user agent may ignore this attribute.

MAXLENGTH=*number*

This attribute specifies the maximum number of characters that can be entered by the user in the text-entry area. The default value for this attribute is an unlimited number of characters.

TITLE=*vdata*

This attribute specifies a title for this element, which may be used in the presentation of this object.

Attributes Defined Elsewhere

The following attribute is defined in section 11.6.1:

TABINDEX

11.6.3.1 INPUT Element Examples

In this example, an INPUT element is specified. This element accepts any characters and displays the input to the user in a human-readable form. The maximum number of character entered is 32 and the resulting input is assigned to the variable named X.

```
<INPUT KEY="X" TYPE="TEXT" MAXLENGTH="32"/>
```

The following example requests input from the user and assigns the resulting input to the variable NAME. The text field has a default value of "Robert".

```
<INPUT KEY="NAME" TYPE="TEXT" DEFAULT="Robert"/>
```

The following example is a card that prompts the user for a first name, last name and age.

```
<CARD>
   First name: <INPUT TYPE="TEXT" KEY="first"/><BR/>
   Last name: <INPUT TYPE="TEXT" KEY="last"/><BR/>
   Age: <INPUT TYPE="TEXT" KEY="age" FORMAT="*N"/>
</CARD>
```

11.6.4 The FIELDSET Element

```
<!ELEMENT FIELDSET (%fields;)* >
<!ATTLIST FIELDSET
  TITLE           %vdata;      #IMPLIED
  >
```

The FIELDSET element allows the grouping of related fields and text. This grouping provides information to the user agent, allowing the optimising of layout and navigation. FIELDSET elements may nest, providing the user with a means of specifying behaviour across a wide variety of devices. See section 11.5.2 for information on how the FIELDSET element may influence layout and navigation.

Attributes

TITLE=*vdata*

This attribute specifies a title for this element, which may be used in the presentation of this object.

11.6.4.1 FIELDSET Element Examples

The following example specifies a WML deck that requests basic identity and personal information from the user. It is separated into multiple field sets, indicating the preferred field grouping to the user agent.

```
<WML>
  <CARD>
    <DO TYPE="ACCEPT">
      <GO URL="/submit?f=$(fname)&amp;l=$(lname)&amp;s=$(sex)&amp;a=$(age)"/>
    </DO>
    <FIELDSET TITLE="Name">
      First name: <INPUT TYPE="TEXT" KEY="fname" MAXLENGTH="32"/><BR/>
      Last name: <INPUT TYPE="TEXT" KEY="lname" MAXLENGTH="32"/><BR/>
    </FIELDSET>
    <FIELDSET TITLE="Info">
      <SELECT KEY="sex">
        <OPTION VALUE="F">Female</OPTION>
        <OPTION VALUE="M">Male</OPTION>
      </SELECT>
      <BR/>
      Age: <INPUT TYPE="TEXT" KEY="age" FORMAT="*N"/>
    </FIELDSET>
  </CARD>
</WML>
```

11.7 The TIMER Element

```
<!ELEMENT TIMER EMPTY>
<!ATTLIST TIMER
  KEY        NMTOKEN          #IMPLIED
  DEFAULT    %vdata;          #REQUIRED
  >
```

The TIMER element declares a card timer, which exposes a means of processing inactivity or idle time. The timer is initialised and started at card entry and is stopped when the card is exited. Card entry is any task or user action that results in the card being activated, for example, navigating into the card. Card exit is defined as the execution of any task (see sections 9.3 and 12.5). The value of a timer will decrement from the initial value, triggering the delivery of an ONTIMER intrinsic event on transition from a value of one to zero. If the user has not exited the card at the time of timer expiration, an ONTIMER intrinsic event is delivered to the card.

Timer resolution is implementation dependent. The interaction of the timer with the user agent's user interface and other time-based or asynchronous device functionality is implementation dependent. It is an error to have more than one TIMER element in a card.

The TIMER timeout value is specified in units of one-tenth (1/10) of a second. The author should not expect a particular timer resolution and should provide the user with another means to invoke a timer's task. If the value of the timeout is not a positive integral number, the user agent must ignore the TIMER element. A timeout value of zero (0) disables the timer.

Attributes

KEY=*nmtoken*

The KEY attribute specifies the name of the variable to be set with the value of the timer. The KEY variable's value is used to set the timeout period upon timer initialisation. The variable named by the KEY 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 KEY variable is set to a value of "0".

DEFAULT=v*data*

The DEFAULT attribute indicates the default value of the variable named in the KEY attribute. When the timer is initialised and the variable named in the KEY attribute is not set, the KEY variable is assigned the value

specified in the DEFAULT attribute. If the KEY variable already contains a value, the DEFAULT attribute is ignored. If the KEY attribute is not specified, the timeout is always initialised to the value specified in the DEFAULT attribute.

11.7.1 TIMER Example

The following deck will display a text message for approximately 10 seconds and will then go to the URL /next.

```
<WML>
   <CARD ONTIMER="/next">
        <TIMER DEFAULT="100"/>
        Hello World!
   </CARD>
</WML>
```

The same example could be implemented as:

```
<WML>
   <CARD>
      <ONEVENT TYPE="ONTIMER">
         <GO URL="/next"/>
      </ONEVENT>
      <TIMER DEFAULT="100"/>
      Hello World!
   </CARD>
</WML>
```

The following example illustrates how a timer can initialise and reuse a counter. Each time the card is entered, the timer is reset to value of the variable t. If t is not set, the timer is set to a value of 5 seconds.

```
<WML>
   <CARD ONTIMER="/next">
      <TIMER KEY="t" DEFAULT="50"/>
      Hello World!
   </CARD>
</WML>
```

11.8 Text

This section defines the elements and constructs related to text.

11.8.1 White Space

WML white space and line break handling is based on [XML] and assumes the default white space handling rules. The WML user agent ignores all *insignificant* white space, as defined by the XML specification. In addition, all other sequences of white space must be compressed into a single inter-word space.

User agents should treat inter-word spaces in a locale-dependent manner, as different written languages treat inter-word spacing in different ways.

11.8.2 Emphasis

```
<!ELEMENT EM     (%flow;)*>
<!ELEMENT STRONG (%flow;)*>
<!ELEMENT B      (%flow;)*>
<!ELEMENT I      (%flow;)*>
<!ELEMENT U      (%flow;)*>
<!ELEMENT BIG    (%flow;)*>
<!ELEMENT SMALL  (%flow;)*>
```

The emphasis elements specify text emphasis markup information.

EM:
Render with emphasis.
STRONG:
Render with strong emphasis.
I:
Render with an italic font.
B:
Render with a bold font.
U:
Render with underline.
BIG:
Render with a large font.
SMALL:
Render with a small font.

Authors should use the STRONG and EM elements where possible. B, I and U elements should not be used except where explicit control over text presentation is required.

11.8.3 Line Breaks

```
<!ENTITY % TAlign   "(LEFT|RIGHT|CENTER)" >
<!ENTITY % BRMode    "(WRAP|NOWRAP)" >

<!ELEMENT BR EMPTY>
<!ATTLIST BR
  ALIGN   %TAlign;   "LEFT"
  MODE    %BRMode;   #IMPLIED
  >
```

WML has two line-wrapping modes: breaking and non-breaking. In breaking mode, line breaks should be inserted into a text flow as appropriate for presentation on an individual device and any inter-word space is a legal line break point. In non-breaking mode, a line of text must not be automatically wrapped.

The non-breaking space entity (` ` or ` `) indicates a space that must not be treated as an inter-word space by the user agent. Authors should use ` ` to prevent undesired line-breaks. The soft-hyphen character entity (`&shy;` or `&#173;`) indicates a location that may be used by the user agent for a line break. If a line break occurs at a soft-hyphen, the user agent must insert a hyphen character (`&#45;`) at the end of the line. In all other operations, the soft-hyphen entity should be ignored. A user agent may choose to entirely ignore soft-hyphens when formatting text lines.

The BR element establishes the beginning of a new line and specifies the line break and alignment parameters for the new line. If the line break mode is not specified, it is identical to the line break mode of the previous line in the current card. If the text alignment is not specified, it defaults to LEFT.

A WML card has a line-break and alignment mode. The initial line break mode for a card is `MODE="WRAP"` (breaking mode) and the initial text alignment is `ALIGN="LEFT"` (left alignment). If the first non-whitespace markup in a card is a BR element, the BR begins the first line in the card. If the first non-whitespace markup in a card is not a BR element, a new line is implicitly started with the default line break and alignment modes.

The treatment of a line too long to fit on the screen is specified by the current line-break mode. If `MODE="WRAP"` is specified, the line is word-wrapped onto multiple lines. If `MODE="LINE"` is specified, the line is not wrapped. The user agent must provide a mechanism to view entire non-wrapped lines (eg, horizontal scrolling or some other user-agent-specific mechanism).

Attributes

ALIGN=*(LEFT|RIGHT|CENTER)*

This attribute specifies the text alignment mode for the line. Text can be centre aligned, left aligned or right aligned when it is displayed to the user. Left alignment is the default alignment mode. If not explicitly

specified, the text alignment is set to the default alignment. For example, a simple <BR/> element starts a new line and sets the alignment to LEFT.

MODE=*(WRAP|NOWRAP)*

This attribute specifies the line-breaking mode for the subsequent text line. WRAP specifies breaking text mode and NOWRAP specifies non-breaking text mode. If not explicitly specified, the line-break mode is identical to the line-break mode of the previous line in the text flow. For example, a simple <BR/> element starts a new line, but does not change the current line-break mode.

11.8.3.1 Line Break Examples

The following example demonstrates how the BR element affects text alignment and line break mode.

```
<WML>
   <CARD>
      line 1, three-line card      <!-- left alignment, breaking mode -->
      <BR ALIGN="RIGHT"/>line 2    <!-- right alignment, breaking mode -->
      <BR MODE="NOWRAP"/>line 3    <!-- left alignment, non-breaking mode -->
   </CARD>
   <CARD>
      <BR ALIGN="CENTER"/>
      line 1, one-line card        <!-- centre alignment, breaking mode -->
   </CARD>
   <CARD>
      <BR MODE="NOWRAP"/>
      <BR ALIGN="CENTER"/>
      line 2, two-line card        <!-- centre alignment, non-breaking mode -->
   </CARD>
</WML>
```

The following example demonstrates a more complex card and the interaction text alignment and line break modes.

```
<WML>
   <CARD>
      <FIELDSET>
         line 1                       <!-- left alignment, breaking mode -->
         <BR ALIGN="NOWRAP"/>line 2   <!-- left alignment, non-breaking mode -->
         <BR MODE="RIGHT"/>line 3     <!-- right alignment, non-breaking mode -->
      </FIELDSET>
      <FIELDSET>
         Choose:                      <!-- right alignment, non-breaking mode -->
         <SELECT KEY="X">
            <OPTION VALUE="1">One</OPTION>
            <OPTION VALUE="2">Two</OPTION>
         </SELECT>
         continuation of line 3       <!-- right alignment, non-breaking mode -->
      </FIELDSET>
      <FIELDSET>
         still on line 3              <!-- right alignment, non-breaking mode -->
         <INPUT KEY="Y"/>
         <BR MODE="WRAP"/>line 4      <!-- left alignment, breaking mode -->
      </FIELDSET>
   </CARD>
</WML>
```

11.8.4 The TAB Element

The following elements specify tab columns.

```
<!ENTITY % tab     "TAB">
<!ENTITY % TAlign   "(LEFT|RIGHT|CENTER)" >
```

```
<!ELEMENT TAB EMPTY>
<!ATTLIST TAB
  ALIGN   %TAlign;   "LEFT"
  >
```

The TAB element is used to create aligned columns in a line. TAB elements separate markup into columns, but do not specify column widths. Columns are specified in row order and a given row is terminated by a BR or any other non-`%text;` element that terminates a line.

A column group is defined as the largest set of contiguous lines containing TAB elements that can be formed at any given point in the text flow. Depending on the display characteristics, the user agent may create aligned columns for each column group, or may use a single set of aligned columns for all column groups in a card. To ensure the narrowest display width, the user agent should determine the width of each column from the maximum width of the text and images in that column. A non-zero width gutter must be used to separate each non-empty column.

Columns empty in all lines of a column group may be ignored. A given line may have fewer TAB elements than other lines in its column group, in which case its right hand columns are assumed empty.

Attributes

`ALIGN=`*`(LEFT|RIGHT|CENTER)`*

This attribute specifies the text layout within a column. Text can be centre aligned, left aligned or right aligned when it is displayed to the user. Left alignment is the default.

11.8.5 TAB Examples

The following example contains a card with a single column group, containing two columns and three rows.

```
<WML>
   <CARD>
      One <TAB/> Two <BR/>
      1   <TAB/>     <BR/>
      A   <TAB/> B   <BR/>
   </CARD>
</WML>
```

An acceptable layout for this card is:

```
  One    Two
  1
  A      B
```

The following example contains a card with two column groups, separated by a line of text. This example demonstrates that multiple column groups may be rendered with separate column widths.

```
<WML>
   <CARD>
      alpha <TAB/> beta <BR/>
      gamma <TAB/> epsilon <BR/>

      this is a test<BR/>

      1 <TAB/> 2 <BR/>
      3 <TAB/> 4 <BR/>
   </CARD>
</WML>
```

An acceptable layout for this card is:

```
  alpha  beta
  gamma  epsilon
  this is a test…
  1  2
```

```
   3  4
```

The following example contains a card with one column group, a variety of column alignments, and an empty initial column.

```
<WML>
   <CARD>
      <TAB/>                 alpha  <TAB/>                 beta    <BR/>
      <TAB/>                 gamma  <TAB/>                 epsilon <BR/>
      <TAB/>                        <TAB ALIGN="RIGHT"/>   2       <BR/>
      <TAB ALIGN="CENTER"/>  3      <TAB ALIGN="CENTER"/> 4       <BR/>
   </CARD>
</WML>
```

An acceptable layout for this card is:

```
   alpha   beta
   gamma   epsilon
                 2
     3         4
```

11.9 Images

```
<!ENTITY % IAlign "(TOP|MIDDLE|BOTTOM)" >
<!ELEMENT IMG EMPTY>
<!ATTLIST IMG
  ALT        %vdata;      #IMPLIED
  SRC        %URL;        #IMPLIED
  LOCALSRC   %vdata;      #IMPLIED
  VSPACE     %length;     "0"
  HSPACE     %length;     "0"
  ALIGN      %IAlign;     "BOTTOM"
  HEIGHT     %length;     #IMPLIED
  WIDTH      %length;     #IMPLIED
  >
```

The IMG element indicates that an image is to be included in the text flow. Image layout is done within the context of normal text layout.

Attributes

ALT=*vdata*

This attribute specifies an alternative textual representation for the image. This representation is used when the image can not be displayed using any other method (ie, the user agent does not support images, or the image contents can not be found).

SRC=*URL*

This attribute specifies the URL for the image. If the browser supports images, it downloads the image from the specified URL and renders it when the text is being displayed.

LOCALSRC=*vdata*

This attribute specifies an alternative internal representation for the image. This representation is used if it exists; otherwise the image is downloaded from the URL specified in the SRC attribute, ie, any LOCALSRC parameter specified takes precedence over the image specified in the SRC parameter.

VSPACE=*length*
HSPACE=*length*

These attributes specify the amount of white space to be inserted to the left and right (HSPACE) and above and below (VSPACE) an image or object. The default value for this attribute is not specified, but is generally a small, non-zero length. If `length` is specified as a percentage value, the resulting size is based on the available horizontal or vertical space, not on the natural size of the image. These attributes are hints to the user agent and may be ignored.

ALIGN=*(TOP|MIDDLE|BOTTOM)*

This attribute specifies image alignment within the text flow and with respect to the current insertion point. ALIGN has three possible values:

- BOTTOM: means that the bottom of the image should be vertically aligned with the current baseline. This is the default value.
- MIDDLE: means that the centre of the image should be vertically aligned with the centre of the current text line.
- TOP: means that the top of the image should be vertically aligned with the top of the current text line.

HEIGHT=*length*
WIDTH=*length*

These attributes give user agents an idea of the size of an image or object so that they may reserve space for it and continue rendering the card while waiting for the image data. User agents may scale objects and images to match these values if appropriate. If length is specified as a percentage value, the resulting size is based on the available horizontal or vertical space, not on the natural size of the image. These attributes are a hint to the user agent and may be ignored.

12. User Agent Semantics

12.1 Deck Access Control

The introduction of variables into WML exposes potential security issues that do not exist in other markup languages such as HTML. In particular, certain variable state may be considered private by the user. While the user may be willing to send a private information to a secure service, an insecure or malicious service should not be able to retrieve that information from the user agent by other means.

A conforming WML user agent must implement deck-level access control, including the `ACCESS` element and the `PUBLIC`, `SENDREFERER`, `DOMAIN` and `PATH` attributes.

A WML author should remove private or sensitive information from the browser context by clearing the variables containing this information.

12.2 Low-Memory Behaviour

WML is targeted at devices with limited hardware resources, including significant restrictions on memory size. It is important that the author have a clear expectation of device behaviour in error situations, including those caused by lack of memory.

12.2.1 Limited History

The user agent may limit the size of the history stack (ie, the depth of the historical navigation information). In the case of history size exhaustion, the user agent should delete the least-recently-used history information.

It is recommended that all user agents implement a minimum history stack size of ten entries.

12.2.2 Limited Browser Context Size

In some situations, it is possible that the author has defined an excessive number of variables in the browser context, leading to memory exhaustion.

In this situation, the user agent should attempt to acquire additional memory by reclaiming cache and history memory as described in sections 12.2.1. If this fails and the user agent has exhausted all memory, the user should be notified of the error, and the user agent should be reset to a predictable user state. For example, the browser may be terminated or the context may be cleared and the browser reset to a well-known state.

12.3 Error Handling

Conforming user agents must enforce error conditions defined in this specification and must not hide errors by attempting to infer author or origin server intent.

12.4 Unknown DTD

A WML deck encoded with an alternate DTD may include elements or attributes that are not recognised by certain user agents. In this situation, a user agent should render the deck as if the unrecognised tags and attributes were not present. Content contained in unrecognised elements should be rendered.

12.5 Reference Processing Behaviour - Inter-card Navigation

The following process describes the reference model for inter-card traversal in WML. All user agents must implement this process, or one that is indistinguishable from it.

12.5.1 The GO Task

The process of executing a GO task comprises the following steps:

1. If the originating task contains VAR elements, the variable name and value in each VAR 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 10.3 for more information on variable substitution.
2. The target URL is identified and fetched by the user agent. The URL attribute value is converted into a simple string by substituting all referenced variables.
3. The access control parameters for the fetched deck are processed as specified in section 11.3.1.
4. The destination card is located using the fragment name specified in the URL.
    a) If no fragment name was specified as part of the URL, the first card in the deck is the destination card.
    b) If a fragment name was identified and a card has a NAME attribute that is identical to the fragment name, then that card is the destination card.
    c) If the fragment name can not be associated with a specific card, the first card in the deck is the destination card.
5. If the destination card contains a NEWCONTEXT attribute, the current browser context is re-initialised as described in section 10.2.
6. The variable assignments resulting from the processing done in step #1 (the VAR element) are applied to the current browser context.
7. The destination card is pushed onto the history stack.
8. If the destination card specifies an ONENTERFORWARD intrinsic event binding, the task associated with the event binding is executed and processing stops. See section 9.8 for more information.
9. If the destination card contains a TIMER element, the timer is started as specified in section 11.7.
10. The destination card is displayed using the current variable state and processing stops.

12.5.2 The PREV Task

The process of executing a PREV task comprises the following steps:

1. If the originating task contains VAR elements, the variable name and value in each VAR 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 10.3 for more information on variable substitution.
2. The target URL is identified and fetched by the user agent. The history stack is popped and the target URL is the top of the history stack. If there is no previous card in the history stack, processing stops.
3. The access control parameters for the fetched deck are processed as specified in section 11.3.1.
4. The destination card is located using the fragment name specified in the URL.
    a) If no fragment name was specified as part of the URL, the first card in the deck is the destination card.
    b) If a fragment name was identified and a card has a NAME attribute that is identical to the fragment name, then that card is the destination card.
5. The variable assignments resulting from the processing done in step #1 (the VAR element) are applied to the current browser context.

6. If the destination card specifies an ONENTERBACKWARD intrinsic event binding, the task associated with the event binding is executed and processing stops. See section 9.8 for more information.
7. If the destination card contains a TIMER element, the timer is started as specified in section 11.7.
8. The destination card is displayed using the current variable state and processing stops.

12.5.3 The NOOP Task

No processing is done for a NOOP task.

12.5.4 The REFRESH Task

The process of executing a REFRESH task comprises the following steps:

1. For each VAR element, the variable name and value in each VAR 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 10.3 for more information on variable substitution.
2. The variable assignments resulting from the processing done in step #1 (the VAR element) are applied to the current browser context.
3. The current card is re-displayed using the current variable state and processing stops.

12.5.5 Task Execution Failure

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

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

13. WML Reference Information

WML is an application of [XML] version 1.0.

13.1 Document Identifiers

Ed: these identifiers have not yet been registered with the IANA or ISO 9070 Registrar

13.1.1 SGML Public Identifier

```
-//WAPFORUM//DTD WML 1.0//EN
```

13.1.2 WML Media Type

Textual form:

```
text/x-wap.wml
```

Tokenized form:

```
application/x-wap.wmlc
```

Ed: these types are not yet registered with the IANA and are consequently *experimental* media types.

13.2 Document Type Definition (DTD)

```
<!--
Wireless Markup Language (WML) Document Type Definition.
WML is an XML language.  Typical usage:
   <?xml version="1.0"?>
   <!DOCTYPE WML PUBLIC "-//WAPFORUM//DTD WML 1.0//EN"
                        "http://www.wapforum.org/DTD/wml.xml">
   <WML>
   ...
   </WML>
-->

<!ENTITY % length  "CDATA">    <!-- [0-9]+ for pixels or [0-9]+"%" for
                                    percentage length -->
<!ENTITY % vdata   "CDATA">    <!-- attribute value possibly containing
                                    variable references -->
<!ENTITY % URL     "%vdata;">  <!-- URL or URN designating a hypertext
                                    node. May contain variable references -->
<!ENTITY % boolean "(TRUE|FALSE)">
<!ENTITY % number  "NMTOKEN">  <!-- a number, with format [0-9]+ -->

<!ENTITY % emph    "EM | STRONG | B | I | U | BIG | SMALL">
<!ENTITY % tab     "TAB">
<!ENTITY % layout  "BR">

<!ENTITY % text     "#PCDATA | %emph; | %tab;">
<!ENTITY % inline   "%text; | %layout;">

<!-- flow covers "card-level" elements, such as text and images -->
<!ENTITY % flow     "%inline; | IMG | A">

<!-- Task types -->
<!ENTITY % task   "GO | PREV | NOOP | REFRESH">

<!-- Navigation and event elements -->
<!ENTITY % navelmts "DO | ONEVENT">

<!--================ Decks and Cards ================-->

<!ELEMENT WML ( HEAD?, TEMPLATE?, CARD+ )>
<!ATTLIST WML
  xml:lang        NMTOKEN        #IMPLIED
  >

<!-- card intrinsic events -->
<!ENTITY % cardev
 "ONENTERFORWARD  %URL;          #IMPLIED
  ONENTERBACKWARD %URL;          #IMPLIED
  ONTIMER         %URL;          #IMPLIED"
  >

<!-- CARD field types -->
<!ENTITY % fields  "%flow; | INPUT | SELECT | FIELDSET">
```

```
<!ELEMENT CARD (%fields; | %navelmts; | TIMER)*>
<!ATTLIST CARD
  NAME            NMTOKEN       #IMPLIED
  TITLE           %vdata;       #IMPLIED
  NEWCONTEXT      %boolean;     "FALSE"
  STYLE           (LIST|SET)    "LIST"
  %cardev;
  >

<!--================ Event Bindings ================-->

<!ELEMENT DO (%task;)>
<!ATTLIST DO
  TYPE         CDATA       #REQUIRED
  LABEL        %vdata;     #IMPLIED
  NAME         NMTOKEN     #IMPLIED
  OPTIONAL     %boolean;   "FALSE"
  >

<!ELEMENT ONEVENT (%task;)>
<!ATTLIST ONEVENT
  TYPE         CDATA       #REQUIRED
  >

<!--================ Deck-level declarations ================-->

<!ELEMENT HEAD ( ACCESS | META )+>

<!ELEMENT TEMPLATE (%navelmts;)*>
<!ATTLIST TEMPLATE
  %cardev;
  >

<!ELEMENT ACCESS EMPTY>
<!ATTLIST ACCESS
  DOMAIN       CDATA       #IMPLIED
  PATH         CDATA       #IMPLIED
  PUBLIC       %boolean;   "FALSE"
  >

<!ELEMENT META EMPTY>
<!ATTLIST META
  HTTP-EQUIV     CDATA      #IMPLIED
  NAME           CDATA      #IMPLIED
  USER-AGENT     CDATA      #IMPLIED
  CONTENT        CDATA      #REQUIRED
  SCHEME         CDATA      #IMPLIED
  >
```

```
<!--================ Tasks ================-->

<!ELEMENT GO (VAR)*>
<!ATTLIST GO
  URL             %URL;           #REQUIRED
  SENDREFERER     %boolean;       "FALSE"
  METHOD          (POST|GET)      "GET"
  ACCEPT-CHARSET  CDATA           #IMPLIED
  POSTDATA        %vdata;         #IMPLIED
  >

<!ELEMENT PREV (VAR)*>

<!ELEMENT REFRESH (VAR)+>

<!ELEMENT NOOP EMPTY>

<!--================ VAR ================-->

<!ELEMENT VAR EMPTY>
<!ATTLIST VAR
  NAME         %vdata;           #REQUIRED
  VALUE        %vdata;           #REQUIRED
  >

<!--================ CARD Fields ================-->

<!ELEMENT SELECT (OPTGROUP|OPTION)+>
<!ATTLIST SELECT
  TITLE        %vdata;           #IMPLIED
  KEY          NMTOKEN           #IMPLIED
  DEFAULT      %vdata;           #IMPLIED
  IKEY         NMTOKEN           #IMPLIED
  IDEFAULT     %vdata;           #IMPLIED
  MULTIPLE     %boolean;         "FALSE"
  TABINDEX     %number;          #IMPLIED
  >

<!ELEMENT OPTGROUP (OPTGROUP|OPTION)+ >
<!ATTLIST OPTGROUP
  TITLE      %vdata;    #IMPLIED
  >

<!ELEMENT OPTION (%text; | ONEVENT)*>
<!ATTLIST OPTION
  VALUE      %vdata;    #IMPLIED
  TITLE      %vdata;    #IMPLIED
  ONCLICK    %URL;      #IMPLIED
  >
```

```
<!ELEMENT INPUT EMPTY>
<!ATTLIST INPUT
  KEY        NMTOKEN           #REQUIRED
  TYPE       (TEXT|PASSWORD)   "TEXT"
  VALUE      %vdata;           #IMPLIED
  DEFAULT    %vdata;           #IMPLIED
  FORMAT     CDATA             #IMPLIED
  EMPTYOK    %boolean;         "FALSE"
  SIZE       %number;          #IMPLIED
  MAXLENGTH  %number;          #IMPLIED
  TABINDEX   %number;          #IMPLIED
  TITLE      %vdata;           #IMPLIED
  >

<!ELEMENT FIELDSET (%fields;)* >
<!ATTLIST FIELDSET
  TITLE          %vdata;      #IMPLIED
  >

<!ELEMENT TIMER EMPTY>
<!ATTLIST TIMER
  KEY        NMTOKEN           #IMPLIED
  DEFAULT    %vdata;           #REQUIRED
  >

<!--================ Images ================-->

<!ENTITY % IAlign "(TOP|MIDDLE|BOTTOM)" >

<!ELEMENT IMG EMPTY>
<!ATTLIST IMG
  ALT        %vdata;      #IMPLIED
  SRC        %URL;        #IMPLIED
  LOCALSRC   %vdata;      #IMPLIED
  VSPACE     %length;     "0"
  HSPACE     %length;     "0"
  ALIGN      %IAlign;     "BOTTOM"
  HEIGHT     %length;     #IMPLIED
  WIDTH      %length;     #IMPLIED
  >

<!--================ Anchor ================-->

<!ELEMENT A ( %inline; | GO | PREV | REFRESH )*>
<!ATTLIST A
  TITLE        %vdata;      #IMPLIED
  >

<!--================ Text layout and line breaks ================-->

<!-- Text alignment attributes -->
<!ENTITY % TAlign   "(LEFT|RIGHT|CENTER)" >

<!ELEMENT TAB EMPTY>
<!ATTLIST TAB
  ALIGN   %TAlign;   "LEFT"
  >
```

```
<!ELEMENT EM     (%flow;)*>
<!ELEMENT STRONG (%flow;)*>
<!ELEMENT B      (%flow;)*>
<!ELEMENT I      (%flow;)*>
<!ELEMENT U      (%flow;)*>
<!ELEMENT BIG    (%flow;)*>
<!ELEMENT SMALL  (%flow;)*>

<!ENTITY % BRMode    "(WRAP|NOWRAP)" >
<!ELEMENT BR EMPTY>
<!ATTLIST BR
  ALIGN   %TAlign;   "LEFT"
  MODE    %BRMode;   #IMPLIED
  >

<!ENTITY quot  "&#34;">     <!-- quotation mark -->
<!ENTITY amp   "&#38;#38;"> <!-- ampersand -->
<!ENTITY apos  "&#39;">     <!-- apostrophe -->
<!ENTITY lt    "&#38;#60;"> <!-- less than -->
<!ENTITY gt    "&#62;">     <!-- greater than -->
<!ENTITY nbsp  " ">    <!-- non-breaking space -->
<!ENTITY shy   "&#173;">    <!-- soft hyphen (discretionary hyphen) -->
```

14. A Compact Binary Representation of WML

WML may be encoded using a compact binary representation. This content format is based upon the WAP Binary XML Content Format [WBXML].

14.1 Extension Tokens

14.1.1 Global Extension Tokens

The [WBXML] global extension tokens are used to represent WML variables. Variable references may occur in a variety of places in a WML deck (see section 10.3). There are several codes that indicate variable substitution. Each code has different escaping semantics (eg, direct substitution, escaped substitution and unescaped substitution). The variable name is encoded in the current document character encoding and must be encoded as the specified in the source document (eg, variable names may not be shortened, mapped or otherwise changed). For example, the global extension token `EXT_I_0` represents an escaped variable substitution, with the variable name inline.

14.1.2 Tag Tokens

WML defines a set of single-byte tokens corresponding to the tags defined in the DTD. All of these tokens are defined within code page zero.

14.1.3 Attribute Tokens

WML defines a set of single-byte tokens corresponding to the attribute names and values defined in the DTD. All of these tokens are defined within code page zero.

14.2 Encoding Semantics

14.2.1 Encoding Variables

All variable references must be converted to variable reference tokens (eg, `EXT_I_0`).

14.2.2 Document Validation

XML document validation (see [XML]) should occur during the process of tokenizing a WML deck and must be based on the DOCTYPE declared in the WML deck. When validating the source text, the tokenization process must accept any DOCTYPE or public identifier, if the document is identified as a WML media type (see section 13.1.2).

The tokenization process should notify the user of any well-formedness or validity errors detected in the source deck.

14.2.2.1 Validate %length;

The WML tokenization process should validate that attribute values defined as `%length;` contain either a `NMTOKEN` or a `NMTOKEN` followed by a percentage sign character. For example, the following attributes are legal:

```
VSPACE="100%"
HSPACE="123"
```

`%length;` data is encoded using normal attribute value encoding methods.

14.2.2.2 Validate %vdata;

The WML tokenization process should validate that attribute values defined as `%vdata;` contain variables and that other `CDATA` attribute values do not. Attribute values not defined in the DTD must allow variable references.

14.3 Numeric Constants

14.3.1 WML Extension Token Assignment

The following global extension tokens are used in WML and occupy document-type-specific token slots in the global token range. As with all tokens in the global range, these codes must be reserved in every code page. All numbers are in hexadecimal.

Table 4. Global extension token assignments

Token Name	***Token***	***Description***
EXT_I_0	40	Variable substitution - escaped. Name of the variable is inline and follows the token as a `termstr`.
EXT_I_1	41	Variable substitution - unescaped. Name of the variable is inline and follows the token as a `termstr`.
EXT_I_2	42	Variable substitution - no transformation. Name of the variable is inline and follows the token as a `termstr`.
EXT_T_0	80	Variable substitution - escaped. Variable name encoded as a reference into the string table.
EXT_T_1	81	Variable substitution - unescaped. Variable name encoded as a reference into the string table.
EXT_T_2	82	Variable substitution - no transformation. Variable name encoded as a reference into the string table.
EXT_0	C0	Reserved for future use.
EXT_1	C1	Reserved for future use.
EXT_2	C2	Reserved for future use.

14.3.2 Tag Tokens

The following token codes represent tags in code page zero (0). All numbers are in hexadecimal.

Table 5. Tag tokens

Tag Name	***Token***
A	22
ACCESS	23
B	24
BIG	25
BR	26
CARD	27
DO	28
EM	29

Tag Name	***Token***
FIELDSET	2A
GO	2B
HEAD	2C
I	2D
IMG	2E
INPUT	2F
META	30
NOOP	31

Tag Name	***Token***
PREV	32
ONEVENT	33
OPTGROUP	34
OPTION	35
REFRESH	36
SELECT	37
SMALL	38

Tag Name	***Token***
STRONG	39
TAB	3A
TEMPLATE	3B
TIMER	3C
U	3D
VAR	3E
WML	3F

14.3.3 Attribute Start Tokens

The following token codes represent the start of an attribute in code page zero (0). All numbers are in hexadecimal.

Table 6. Attribute start tokens

Attribute Name	***Attribute Value Prefix***	***Token***
ACCEPT-CHARSET		5
ALIGN	BOTTOM	6
ALIGN	CENTER	7
ALIGN	LEFT	8
ALIGN	MIDDLE	9
ALIGN	RIGHT	A
ALIGN	TOP	B
ALT		C
CONTENT		D
DEFAULT		E
DOMAIN		F
EMPTYOK	FALSE	10
EMPTYOK	TRUE	11
FORMAT		12
HEIGHT		13
HSPACE		14
IDEFAULT		15
IKEY		16
KEY		17
LABEL		18
LOCALSRC		19
MAXLENGTH		1A
METHOD	GET	1B

Attribute Name	***Attribute Value Prefix***	***Token***
METHOD	POST	1C
MODE	NOWRAP	1D
MODE	WRAP	1E
MULTIPLE	FALSE	1F
MULTIPLE	TRUE	20
NAME		21
NEWCONTEXT	FALSE	22
NEWCONTEXT	TRUE	23
ONCLICK		24
ONENTERBACKWARD		25
ONENTERFORWARD		26
ONTIMER		27
OPTIONAL	FALSE	28
OPTIONAL	TRUE	29
PATH		2A
POSTDATA		2B
PUBLIC	FALSE	2C
PUBLIC	TRUE	2D
SCHEME		2E
SENDREFERER	FALSE	2F
SENDREFERER	TRUE	30
SIZE		31
SRC		32

Attribute Name	***Attribute Value Prefix***	***Token***
STYLE	LIST	33
STYLE	SET	34
TABINDEX		35
TITLE		36
TYPE		37
TYPE	ACCEPT	38
TYPE	DELETE	39
TYPE	HELP	3A
TYPE	PASSWORD	3B
TYPE	ONCLICK	3C
TYPE	ONENTERBACKWARD	3D
TYPE	ONENTERFORWARD	3E

Attribute Name	***Attribute Value Prefix***	***Token***
TYPE	ONTIMER	3F
TYPE	OPTIONS	45
TYPE	PREV	46
TYPE	RESET	47
TYPE	TEXT	48
TYPE	vnd.	49
URL		4A
URL	http://	4B
URL	https://	4C
USER-AGENT		4D
VALUE		4E
VSPACE		4F
WIDTH		50
xml:lang		51

14.3.4 Attribute Value Tokens

The following token codes represent attribute values in code page zero (0). All numbers are in hexadecimal.

Table 7. Attribute value tokens

Attribute Value	***Token***
.com/	85
.edu/	86
.net/	87
.org/	88
ACCEPT	89
BOTTOM	8A
CLEAR	8B
DELETE	8C
HELP	8D
http://	8E
http://www.	8F
https://	90
https://www.	91
LIST	92
MIDDLE	93

Attribute Value	***Token***
NOWRAP	94
ONCLICK	95
ONENTERBACKWARD	96
ONENTERFORWARD	97
ONTIMER	98
OPTIONS	99
PASSWORD	9A
RESET	9B
SET	9C
TEXT	9D
TOP	9E
UNKNOWN	9F
WRAP	A0
www.	A1

14.4 WML Encoding Examples

Refer to [WBXML] for additional examples.

The following is another example of a tokenized WML deck. It demonstrates variable encoding, attribute encoding and the use of the string table. Source deck:

```
<WML>
  <CARD NAME="abc" STYLE="LIST">
    <DO TYPE="ACCEPT">
      <GO URL="http://xyz.org/s"/>
    </DO>
    X: $(X)<BR/>
    Y: $(&#x59;)<BR MODE="NOWRAP"/>
    Enter name: <INPUT TYPE="TEXT" KEY="N"/>
  </CARD>
</WML>
```

Tokenized form (numbers in hexadecimal) follows. This example only uses inline strings and assumes that the character encoding uses a NULL terminated string format. It also assumes that the character encoding is UTF-8:

```
00  02  04  'X' 00  'Y' 00  7F  E8  21  03  'a' 'b' 'c' 00
33  01  E9  38  01  AD  4B  03  'x' 'y' 'z' 00  88  03  's'
00  01  03  ' ' 'X' ':' ' ' 00  82  00  27  03  ' ' 'Y' ':'
' ' 00  82  02  A7  1D  01  03  ' ' 'E' 'n' 't' 'e' 'r' ' '
'n' 'a' 'm' 'e' ':' ' ' 00  B1  48  18  03  'N' 00  01  01
01
```

In an expanded and annotated form:

Table 8. Example tokenized deck

Token Stream	***Description***
00	WBXML Version number
02	WML Public ID
04	String table length
'X', 00, 'Y', 00	String table
7F	WML, with content
E9	CARD, with content and attributes
21	NAME=
03	Inline string follows
'a', 'b', 'c', 00	string
33	STYLE="LIST"
01	END (of CARD attribute list)
EA	DO, with content and attributes
38	TYPE=ACCEPT
01	END (of DO attribute list)
AD	GO, with attributes
4B	URL="http://"
03	Inline string follows

Token Stream	***Description***
'x', 'y', 'z', 0	string
88	".org/"
03	Inline string follows
's', 0	string
01	END (of DO element)
03	Inline string follows
' ', 'X', ':', ' ', 00	String
82	Direct variable reference (EXT_T_2)
00	Variable offset 0
28	BR
03	Inline string follows
' ', 'Y', ':', ' ', 00	String
82	Direct variable reference (EXT_T_2)
02	Variable offset 2
A7	BR, with attributes
1D	MODE="NOWRAP"
01	END (of BR attribute list)
03	Inline string follows
' ', 'E', 'n', 't', 'e', 'r', ' ', 'n', 'a', 'm', 'e', ':', ' ', 00	String
B1	INPUT, with attributes
48	TYPE="TEXT"
18	KEY=
03	Inline string follows
'N', 00	String
01	END (of INPUT attribute list)
01	END (of CARD element)
01	END (of WML element)